RENATA ABRANTES DE CAMPOS GORINI

UM AMBIENTE DE SUPORTE À AUTORIA COOPERATIVA DE DOCUMENTOS HIPERMÍDIA

DISSERTAÇÃO DE MESTRADO

Departamento de Informática

Rio de Janeiro, 11 de Setembro de 2001.

RENATA ABRANTES DE CAMPOS GORINI

UM AMBIENTE DE SUPORTE À AUTORIA COOPERATIVA DE DOCUMENTOS HIPERMÍDIA

Dissertação de Mestrado apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Mestre em Informática: Ciência da Computação.

Orientador: Luiz Fernando Gomes Soares.

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 11 de setembro de 2001.

Este trabalho é dedicado aos meus pais, Nelson Gorini e Vera Lúcia Abrantes de Campos Gorini, e à minha irmã pelo apoio e incentivo constante.

AGRADECIMENTOS

- Ao professor Luiz Fernando Gomes Soares, meu orientador e mestre, pela competência profissional, pelo exemplo, pela disponibilidade e atenção, e pela confiança que em mim depositou.

- Aos meus pais e irmã, fortemente presentes durante todo o desenvolvimento deste trabalho, pelo constante incentivo e apoio, e, principalmente, por todo amor e carinho.

- Ao André, por seu amor, carinho e compreensão pelos vários momentos roubados da nossa convivência.

- A minhas amigas, que participaram dessa fase, pela presença, palavras de apoio e incentivo, e pela "simples" amizade.

- À equipe do Laboratório TeleMídia, pela amizade e pelo companheirismo, que muito contribuíram para a realização deste trabalho de uma forma mais prazerosa. Em especial, a Débora Muchaluat e Rogério Rodrigues, pela presença constante, apoio e orientação indispensáveis à efetivação deste trabalho.

- A todos os amigos, professores e funcionários do Departamento de Informática da PUC-Rio que colaboraram, de uma forma ou de outra, para a conclusão deste trabalho.

- À CAPES pelo apoio financeiro fornecido durante o curso.

- A Deus, pela vida e saúde, e pela oportunidade de realizar este trabalho.

RESUMO

Esta dissertação descreve como um trabalho cooperativo pode ser realizado segundo uma estrutura de tarefas organizadas, através da qual são definidos certos mecanismos que irão possibilitar a colaboração coordenada entre participantes do trabalho. Uma tarefa irá representar um trabalho cooperativo, e um conjunto estruturado de tarefas um trabalho cooperativo mais complexo.

Baseado na definição de tarefas, foi proposto um ambiente de suporte a autoria cooperativa de documentos hipermídia, que irá permitir a estruturação de um determinado trabalho, a definição de uma seqüência para a execução desse trabalho, como em sistemas clássicos de *workflow*, e a definição de mecanismos de suporte à colaboração, comunicação e coordenação, essenciais durante um processo cooperativo.

Esse ambiente ainda foi especializado e implementado para um sistema particular, o HyperProp, utilizando-se conceitos usuais em sistemas hipermídia para a sua implementação, mais especificamente conceitos do modelo NCM.

Palavras-chave: autoria cooperativa, documentos hipermídia, colaboração, comunicação, coordenação e framework.

ABSTRACT

This thesis describes how a cooperative work can be organized using a task structure. Based on this structure, some mechanisms will be defined in order to support a coordinated collaboration between participants of a work. In this context, a task will represent a cooperative work, while a set of tasks will represent a complex cooperative work.

Based on tasks, an environment was proposed to support the cooperative authoring on hypermedia documents. This environment will allow definitions of a task structure, task relationships (to define rules for the execution of a work, represented by tasks) analogous to *workflow* systems, and the definition of collaboration, communication and coordination mechanisms, which are essentials during cooperative processes.

The environment was specialized and implemented for a particular hypermedia system, called HyperProp, using some hypermedia concepts (nodes and links), more precisely some concepts defined by the NCM model (a hypermedia model).

Key words: cooperative authoring, hypermedia documents, collaboration, communication, coordination, and framework.

CONTEÚDO

LISTA DE FIGURAS

1. INTRODUÇÃO

É um fato bem conhecido que documentos são criados, freqüentemente, por um grupo de pessoas. No caso de grandes documentos, esse fato torna-se ainda mais comum. Portanto, é de grande utilidade que um sistema ofereça recursos para que vários indivíduos, possivelmente, distribuídos em localidades diferentes, possam contribuir na autoria de um documento de maneira adequada e eficiente.

O processo onde vários autores criam e editam um determinado documento é denominado de *autoria cooperativa*. Nesse processo, os vários autores devem trabalhar em cooperação, o que envolve colaboração, comunicação e coordenação.

Colaborar significa contribuir em um trabalho onde vários autores participam de maneira não necessariamente harmoniosa e coordenada. Comunicação é o compartilhamento de informações entre os autores por meio de qualquer tipo de mídia (texto, áudio, vídeo, etc.). Coordenação é o ato de gerenciar a interdependência entre as colaborações individuais, realizadas para se atingir um objetivo. A idéia principal é que a cooperação deve ser realizada através de colaborações coordenadas, onde essa coordenação é auxiliada pela comunicação.

A autoria cooperativa inclui realizações de tarefas cooperadas, evitando que pessoas comecem a se envolver em tarefas conflitantes ou repetidas. Dependendo da atividade e do grupo de pessoas envolvidas no trabalho cooperativo, é necessário identificar o suporte necessário para a coordenação, provendo mecanismos de controle sobre as comunicações, e de monitoramento sobre os documentos envolvidos no trabalho cooperativo.

Nesse sentido, é interessante que exista um ambiente de autoria que possibilite tanto o trabalho individual como o trabalho em grupo, onde se possa estruturar o trabalho de maneira organizada. Esse ambiente deve permitir a divisão de um trabalho complexo em trabalhos mais simples e a sua atribuição a diferentes equipes de trabalho organizadas para

a realização de um objetivo em comum. Esse ambiente deve permitir o compartilhamento coordenado de documentos e outras informações, e a comunicação entre atores interessados e responsáveis pelo trabalho.

A coordenação exerce um papel importante, durante a manipulação de objetos compartilhados pelos membros do grupo durante o processo de autoria cooperativa, à medida que devem ser estabelecidas regras de acesso a objetos compartilhados, como, por exemplo, através de: controle de acesso, controle de concorrência e controle de versão.

A motivação para esse trabalho foi encontrada na necessidade de suporte à autoria cooperativa no sistema HyperProp [SoRoM 2000], desenvolvido no laboratório Telemídia da Puc-Rio. Todavia, inicialmente, será proposto um ambiente genérico para suporte à cooperação entre atores[1], que será, posteriormente, especificado para o sistema particular, HyperProp.

O enfoque desse trabalho será na autoria cooperativa hipermídia, que trata do problema da composição e edição de hiperdocumentos (envolvendo sua estrutura e conteúdo) realizada por vários atores. O modelo, que será proposto e que irá possibilitar o suporte à autoria particular ou cooperativa de documentos, pode ser mapeado em conceitos de nós e relacionamentos entre nós (elos), usuais em sistemas hipermídia, para a elaboração de sua estrutura.

1.1 OBJETIVOS

Um dos objetivos desse trabalho consiste em apresentar um modelo genérico de estruturação de bases de trabalho que possibilitará a colaboração coordenada entre membros de um grupo de trabalho. O modelo dará suporte à autoria cooperativa, possibilitando:

- a definição de uma estrutura para a realização do trabalho que irá permitir:

[1] Atores serão responsáveis por um trabalho cooperativo.

  - a divisão de uma tarefa (representando o trabalho que deve ser realizado) em subtarefas, e as suas atribuições aos participantes de um grupo de trabalho;
  - e a definição de uma ordem para realização das tarefas definidas (através de relacionamentos entre tarefas).
- a existência de um ambiente de trabalho particular e outro global;
- a colaboração coordenada de atividades cooperativas;
- e a comunicação entre os participantes durante todo o processo colaborativo.

O modelo genérico de estruturação de bases de trabalho será, posteriormente, transformado em uma estrutura de objetos do modelo conceitual NCM (Nested Context Model) [Soare 2000] e fará parte do sistema Hyperprop [SoRoM 2000], ambos desenvolvidos no laboratório TeleMídia do Departamento de Informática da PUC-Rio. O NCM define os conceitos estruturais dos dados, os eventos e os relacionamentos entre os dados e também define regras de estruturação e operações sobre os dados para manipulação e atualização das estruturas de hiperdocumentos. O sistema Hyperprop segue uma implementação cliente-servidor, e é baseado no modelo conceitual de dados NCM. O sistema tem como objetivo o tratamento de hiperdocumentos, fornecendo meios para autoria e apresentação de documentos hipermídia com sincronismo temporal e espacial.

É também objetivo desta dissertação especificar um *Controle de Acesso* genérico e adaptável, que irá fornecer suporte à coordenação durante a autoria cooperativa, e implementar o controle de acesso para o NCM de acordo com especificações mais recentes do modelo. Essa implementação será acoplada ao sistema HyperProp, e oferecerá as funcionalidades de controle de permissões e de concorrência sobre documentos.

1.2 ORGANIZAÇÃO DA DISSERTAÇÃO

No Capítulo 2, será apresentado um *modelo de tarefas*, que dará suporte à divisão do trabalho em tarefas e à definição de uma ordem para a execução das mesmas. Esse modelo se propõe a ser um modelo flexível que permita a inserção de novas tarefas e remoção de tarefas durante a realização do trabalho, assim como a alteração do processo de execução de tarefas (alteração da organização de realização do trabalho).

O Capítulo 3 discute como o *modelo de tarefas*, apresentado no capítulo 2, irá possibilitar a realização da autoria de documentos em ambientes particulares e globais, e a colaboração coordenada entre os participantes do trabalho. Nesse capítulo, serão apresentados os mecanismos de colaboração, coordenação e comunicação, essenciais ao processo cooperativo. A partir da definição desses mecanismos, será apresentado o *modelo de cooperação*, baseado do *modelo de tarefas*, que irá possibilitar a cooperação entre membros de um grupo, engajados em um determinado trabalho.

No Capítulo 4, será apresentado o Controle de Acesso, responsável por parte da coordenação do trabalho. O Controle de Acesso irá possibilitar a definição de regras de acesso durante a realização do trabalho sobre documento e tarefas.

No Capítulo 5, o ambiente que irá suportar a autoria cooperativa, apresentado nos Capítulos 2 e 3, será especificado para o HyperProp, utilizando-se o modelo conceitual NCM.

Os trabalhos relacionados serão discutidos no Capítulo 6, e, no Capítulo 7, a dissertação é encerrada com algumas considerações finais que incluem comentários e possíveis trabalhos futuros.

2. UM MODELO PARA AUTORIA

Neste capítulo, um modelo de tarefas para o suporte à autoria cooperativa será apresentado. Inicialmente, na Seção 2.1, alguns conceitos serão introduzidos, assim como certos princípios identificados como necessários para suporte à cooperação. Na Seção 2.2, será apresentado o modelo de tarefas, propriamente dito, que irá possibilitar a divisão de trabalho, sua organização e execução.

2.1 CONCEITOS E PRINCÍPIOS

A autoria cooperativa pode ser realizada de maneira assíncrona ou síncrona, segundo um modo de cooperação *fracamente acoplado* ou *fortemente acoplado*.

A autoria *assíncrona* ocorre quando autores não estão presentes durante todo o processo de cooperação. A *autoria síncrona*, quando membros de um grupo estão conectados ao ambiente de trabalho durante a cooperação. Ainda, quando diversos autores atuam em um dado objeto compartilhado, porém utilizando versões do objeto de trabalho em ambientes individuais, a autoria recebe o nome de *semi-síncrona* [MinMa 1993].

No *modo de cooperação fracamente acoplado*, cada autor possui a sua interface que provê informações consistentes sobre os dados dos objetos compartilhados. Alterações em posições ou tamanho de objetos não são refletidas nas demais interfaces, porém remoção, criação de objetos, assim como alterações em objetos o são. A propagação das modificações nos objetos compartilhados deve ser realizada de acordo com algumas políticas que garantam a consistência e corretude do resultado final do trabalho cooperativo.

Autores realizando um trabalho no *Modo de Cooperação Fortemente Acoplado* recebem as alterações realizadas em objeto através de uma interface que muda de estado dependendo da interação com usuários.

[HaWil 1992] identifica três requisitos essenciais para esse modo de cooperação:

- Todos os autores de uma sessão de trabalho[2] devem visualizar a mesma interface (tamanho e conteúdo idênticos);
- Deve existir um canal de comunicação disponível (normalmente um canal de áudio) para os autores;
- Um *telepointer*[3] compartilhado deve ser utilizado.

Se um autor mudar um objeto de posição em sua visão da interface compartilhada, a nova posição tem de ser refletida instantaneamente nas visões dos outros autores. A idéia é que, realmente, todos os aspectos da interface sejam compartilhados ao máximo entre os autores, como se estivessem trabalhando em um espaço único de cooperação. Esse é o modo de cooperação mais difícil de se oferecer suporte, além de ser o que mais consome recursos computacionais.

Durante o processo de cooperação podem ser identificados três conceitos básicos, conhecidos como os três C's do trabalho cooperativo: a colaboração, a comunicação e a coordenação.

A colaboração é o ato de contribuir para a realização de um trabalho em comum. Para isso, a colaboração envolve mecanismos de gerência de objetos compartilhados, à medida que se deve manter a consistência de um objeto quando um acesso concorrente for realizado, e a gerência de interfaces de usuários.

As colaborações podem ser realizadas de maneira assíncrona, síncrona ou semi-síncrona. Em todas as formas de trabalho, o sistema deve garantir que todos os co-autores possuam conhecimento da participação e das atividades que estão sendo realizadas pelos outros, mesmo que esses estejam em locais geograficamente distribuídos, ou não estejam conectados constantemente ao sistema.

---

[2] Uma sessão de trabalho, nesse caso, caracteriza a realização de um trabalho cooperativo.

[3] É um ponteiro, controlado por um usuário, que aponta para uma determinada área de trabalho.

A comunicação possibilita o intercâmbio de informações entre autores, e pode ser fornecida por um mecanismo interno ao sistema, que possibilite a troca de qualquer tipo de mídia (texto, áudio, vídeo, etc.), ou externo ao sistema, por meio de e-mail, telefone, etc. A comunicação é um fator fundamental para a interação do grupo de trabalho, tanto na realização de um trabalho assíncrono como em um trabalho síncrono ou semi-síncrono. Também dependerá do modo de cooperação, o grau de comunicação entre eles.

Um controle de notificação terá grande importância na comunicação entre autores, à medida que tornará possível a notificação a grupos de trabalho sobre eventos realizados por outros grupos de trabalho ou autores específicos. Junto ao controle de notificações, regras de compartilhamento de informação e anotações exercem a base para a comunicação durante o processo cooperativo.

A coordenação irá gerenciar o processo de colaboração, fornecendo mecanismos que coordenem a comunicação entre os participantes e as atividades sendo realizadas. Também se pode considerar formas de marcar eventos, definir prazos e encontros, e de controlar o acesso às áreas particulares, bem como o uso de dados públicos por pessoas autorizadas. Uma forma de coordenação mais formal, normalmente utilizada para dar suporte a um trabalho mais estruturado[4] e bem definido, é o *workflow*[5]. Apesar de fornecerem mecanismos de coordenação eficazes, já que estruturam a realização e execução do trabalho, sistemas de *workflow* são ferramentas ineficazes para a definição de trabalhos não tão bem estruturados, já que não permitem a interação com o usuário durante o processo de execução do trabalho [WanHa 1998].

---

[4] Um exemplo de tarefa bem estruturada seria um processo de aprovação de viagem (onde todas as tarefas e a ordem de execução são conhecidas). Já, outra não tão bem estruturada, poderia representar a *autoria* de um documento, durante a qual podem ser necessárias a criação e a alteração de partes do processo em tempo de execução.

[5] Um sistema que suporte *workflow* possui um conjunto de regras que descrevem como unidades de trabalho fluem através de um processo.

Dependendo da aplicação, das características do grupo ou suporte tecnológico, existirão formas distintas de trabalho, que acarretarão em diferentes necessidades de coordenação.

Uma das formas de coordenação é a livre, onde não são previstos mecanismos explícitos de coordenação. Outra forma é caracterizada por uma coordenação gerida por indivíduos, cada qual com permissões e privilégios distintos. Nesse caso, um coordenador exerce o papel de *coordenador* e que terá privilégios e permissões distintas dos outros participantes, podendo, por exemplo, organizar a comunicação e a realização de tarefas. Uma forma automática de coordenação pode ser realizada por *agentes* [Ferbe 1999][Weiss 2000], que podem coordenar todo o processo de colaboração, como, por exemplo, propagando mensagens necessárias ao processo e sugerindo ações a serem tomadas pelos participantes do trabalho.

Neste capítulo, será especificado um *modelo de tarefas*[6] que dará suporte a um mecanismo de coordenação adaptativo. O modelo possibilitará a coordenação mais formal, através da definição de uma estrutura para a realização do trabalho, como em sistemas de *workflow*, assim como possibilitará a coordenação mais informal, através da definição de papéis e regras de acesso a objetos.

2.2 MODELO DE TAREFAS

O *modelo de tarefas* irá definir um modelo para a organização de um trabalho segundo uma estrutura hierárquica de tarefas, de maneira a possibilitar a definição de regras para a execução de um trabalho.

Tarefas irão representar a realização de um trabalho por um ator[7] ou por um grupo de atores colaborando para a realização de um objetivo comum. Uma tarefa poderá ser

[6] Tarefas representam a realização de um trabalho definido.

[7] Atores serão responsáveis pela realização de um determinado trabalho.

subdividida em subtarefas, representando uma partição do trabalho que deve ser realizado em *T*, que serão atribuídas a atores responsáveis por suas execuções.

A correta execução do trabalho será garantida pela correta execução de tarefas, que, por sua vez, será definida através de relacionamentos entre tarefas.

Relacionamentos irão determinar relações de causalidade e restrição entre tarefas. De maneira geral, relacionamentos de causalidade definirão relações de causa e efeito (condições/ações), e relacionamentos de restrição serão responsáveis pela definição de relações que, semanticamente, especificam restrições, tais como: uma tarefa deverá terminar sua execução, se e somente se outra tarefa já tiver sua execução terminada.

A Figura 2.1 mostra como seria uma estrutura de tarefas criada para retratar a execução de um trabalho, representado por *T1*.

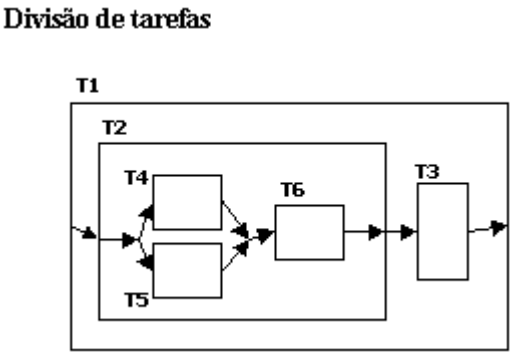


Figura 2.1: Divisão e execução de tarefas.

Relacionamentos poderiam ser definidos entre as tarefas *T2* e *T3* para determinar a execução de *T3* (efeito do relacionamento) após a conclusão de *T2* (condição do relacionamento). Outros poderiam ser definidos entre as tarefas *T4*, *T5* e *T6*, contidas em *T2*, para determinar a execução de *T4* e *T5* paralelamente e a execução de *T6* apenas após a conclusão de *T4* e *T5* (restrição)[8].

[8] É importante notar que, nesse caso, a conclusão de T4 e T5 não necessariamente leva ao início de T6. A restrição é feita apenas ao início de T6.

Mais especificamente, o relacionamento entre tarefas irá determinar o relacionamento entre *eventos* ocorridos em tarefas. Um *evento* pode ser definido como uma ocorrência no tempo, que pode ser instantânea ou durar um período de tempo, e pode representar, por exemplo, o início ou o término de execução de uma determinada tarefa (evento instantâneo), ou a própria execução da tarefa (evento não instantâneo). O evento definido pela realização de uma tarefa será chamado de *evento de autoria*.

Os estados de um evento de autoria representarão os estados possíveis de uma tarefa durante todo o processo de autoria cooperativa. *Eventos de autoria* serão apresentados na Seção 2.2.2.

Para compor o *modelo de tarefas*, devem ser definidos certos atributos relativos a cada tarefa *T*, que serão definidos ao longo deste capítulo, e que possibilitarão a definição de uma estrutura para a realização de um determinado trabalho. Entre esses atributos, estão o ***conjunto de subtarefas*** *(Cst)*, o ***conjunto de relacionamentos*** *(Cr)* e o ***conjunto de documentos*** *(Cd)*. Os dois primeiros irão compor a estrutura de tarefas de *T*, enquanto o último irá representar o trabalho que deve ser realizado em *T*.

Diz-se que uma subtarefa *T1* de *Cst* é um *componente* de *Cst*, e que *T1* está *contida* em *T*. Diz-se também que uma subtarefa *T2* está *recursivamente contida em T*, se e somente se *T2* está contida em *T* ou *T2* está contida em uma subtarefa recursivamente contida em *T*.

O *conjunto de relacionamentos* de *T* irá conter os relacionamentos definidos em *T* que descrevem relações entre subtarefas contidas em *T*, ou entre estas mesmas subtarefas e *T*. Diz-se que um relacionamento *R* de *Cr* é um *componente* de *Cr*, e que *R* está *contido* em *T*.

O *conjunto de documentos* de *T* irá conter documentos utilizados durante o trabalho em *T*. Diz-se que um documento *D* de *Cd* é um *componente* de *Cd* e que *D* está *contido* em *T*. Deve-se acrescentar que, um determinado documento *D* só pode estar contido em apenas uma tarefa de um conjunto de tarefas que representa um trabalho cooperativo.

As definições de tarefas e relacionamentos que organizam o trabalho, que deve ser realizado, caracterizam o processo de *autoria de tarefas*, que como poderá ser visto, é muito semelhante ao processo de *autoria de documentos* (mais precisamente, semelhante a autoria da estrutura de documentos). Em uma tarefa *T*, a *autoria cooperativa de documentos* (em *T*) será realizada por um conjunto de indivíduos, interagindo segundo um modo de cooperação específico, de acordo com determinadas regras que serão definidas ao longo do trabalho. O processo de autoria de documentos poderá ser visto no Capítulo 3.

2.2.1 AUTORIA DE TAREFAS

Para que a estrutura de execução de um determinado trabalho seja criada de acordo com o *modelo de tarefas*, devem ser criados tarefas e relacionamentos entre tarefas. Esta *autoria de tarefas e relacionamentos* será realizada por um conjunto de atores, responsáveis por esta autoria.

Uma *tarefa* possuirá, então, como outro de seus atributos um ***conjunto de atores***, cada qual representando um determinado *papel*. Diz-se que um ator pertence à *T*, se esse pertence ao *conjunto de atores* de *T*.

Papéis determinarão a definição de responsabilidades que devem ser atribuídas a atores.

2.2.1.1 PAPÉIS

A definição de papéis irá possibilitar a coordenação do trabalho durante o processo de colaboração. Através deles poderão ser definidas restrições, não só de direitos de acesso a documentos, como também de direitos de acesso a tarefas. Neste trabalho, dois papéis foram definidos, no entanto outros poderiam ser acrescidos.

O papel de *coordenador* de uma tarefa *T* deve ser atribuído a um ator responsável pela gerência do trabalho realizado em *T* e pela definição da ordem de realização de subtarefas de *T*. O coordenador de uma tarefa *T* terá permissão para:

- criar novas subtarefas contidas em *T*;
- remover subtarefas contidas em *T*;
- criar e remover relacionamentos entre subtarefas contidas em *T*;
- incluir e remover atores no *conjunto de atores* de *T*, e alterar papéis de atores pertencentes à T;
- alterar o estado da tarefa[9] (concluir uma tarefa, suspender uma tarefa...);
- criar e alterar documentos contidos em *T*;
- criar anotações[10] em *T*, sobre *T*, sobre documentos contidos em *T*, ou sobre subtarefas contidas em *T*.

É importante notar que essas permissões são definidas para o *coordenador* de uma tarefa *T*, e, portanto, só serão válidas para esta tarefa *T*, isto é, atores pertencentes à *T* com papel de *coordenador* só terão permissão para realizar estas operações em *T*, e não em outras tarefas ou subtarefas contidas em *T*.

O papel de *resolvedor* de uma tarefa *T* deve ser atribuído a um ator pertencente à *T*, que será responsável pela execução da tarefa, podendo:

- criar e alterar documentos contidos em *T*;
- criar anotações em *T*, sobre *T*, ou sobre documentos contidos em *T*.De maneira geral, esses dois papéis são úteis na divisão básica de responsabilidades durante a autoria de tarefas e documentos, fornecendo meios para a coordenação de tarefas. Todavia, outros papéis, que definissem diferentes restrições de acesso poderiam ser muito importantes. Por isso, a definição de novos papéis dependerá da necessidade do processo de autoria, cabendo ao sistema que irá implementar o modelo proposto tornar esta definição de papéis adaptável a novas situações.

---

[9] Diferentes estados de uma tarefa são representados por diferentes estados do *evento de autoria* definido na Seção 2.2.2.

[10] Anotações serão tratadas na Seção 3.2.

No HyperProp, diferentes papéis poderão ser definidos, como se verá, através do Controle de Acesso, que será especificado no Capítulo 4.

2.2.1.2 CRIAÇÃO DE TAREFAS

Inicialmente, quando uma tarefa *T* é criada, deve ser definido o conjunto de atores que irão realizar algum trabalho em *T*.

Só poderão ser criadas novas subtarefas em uma tarefa *T* por atores de *T* com permissão para criar subtarefas, ou seja, atores com papel de *coordenador*. O cenário descrito aqui utilizará os *papéis* definidos anteriormente, mas deve-se mencionar mais uma vez que outros papéis poderão ser definidos.

⇒ Quando *T* é criada, o *conjunto de atores* de *T* deve conter um único *ator* com papel de *coordenador*, representando o criador da tarefa. Com o papel de *coordenador*, esse ator terá permissão para inserir novos atores no *conjunto de atores* de *T* e criar novas subtarefas em *T*.

⇒ Quando uma subtarefa é definida em *T*, esta subtarefa terá, inicialmente, em seu *conjunto de atores*, o coordenador de *T* que a criou, também com papel de *coordenador*.

Deverá existir pelo menos um *ator* com papel de *coordenador* em cada tarefa *T* (e nada impede que possa ter mais de um ator com papel de *coordenador*) para que seja possível, de maneira geral, a gerência de alterações em *T*. Isso inclui a inserção e remoção de atores no *conjunto de atores* de *T*, a alteração do *estado* de *T*, o que irá possibilitar a conclusão de *T*, e a inserção e remoção de documentos no *conjunto de documentos* de *T*.

2.2.1.3 CRIAÇÃO DE RELACIONAMENTOS

Relacionamentos contidos no *conjunto de relacionamentos* de uma tarefa *T*, só poderão ser criados, assim como subtarefas, por um ator de *T*, com papel de *coordenador*.

A única restrição que deve ser feita à criação de relacionamentos entre tarefas, em *T*, é que esses não poderão formar ciclos de relacionamentos.

Como foi dito anteriormente, relacionamentos entre tarefas irão definir relações entre *eventos de autoria* relativo às tarefas relacionadas. Estados desses eventos irão determinar estados de tarefas para o processo de autoria.

Relacionamentos serão melhor definidos na Seção 2.2.3.

2.2.2 EVENTO DE AUTORIA

Como definido em 2.2, a realização de uma tarefa define um *evento de autoria*. Possíveis estados de um *evento de autoria* representarão os estados possíveis de uma tarefa durante todo o processo de autoria cooperativa.

De maneira geral, um *evento de autoria* pode estar nos seguintes estados: *dormindo*, *preparando*, *preparado*, *ocorrendo*, *suspenso*, *abortado* e *concluído*. O sistema que irá controlar a execução do trabalho irá se basear em uma máquina de estados, própria ao *evento de autoria*, e irá gerir as possíveis transições de estado durante a autoria cooperativa de documentos.

A figura seguinte apresenta, de forma resumida, as fases do trabalho realizado em uma tarefa. A máquina de estados completa do *evento de autoria* pode ser vista na Figura 2.3.

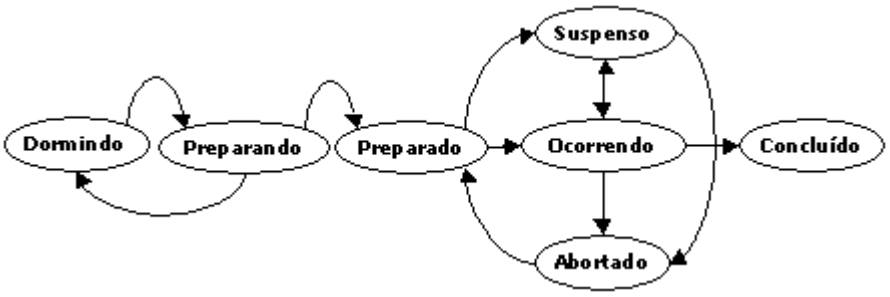


Figura 2.2: Fases do processo de autoria.

Inicialmente, quando uma tarefa *T* está sendo definida, o *evento de autoria* relativo à *T* está no estado *dormindo*, e permanece assim durante a fase de criação da tarefa.

Para que a tarefa *T* seja executada, o *evento de autoria* deve passar para o estado *preparando* e em seguida para o estado *preparado*. Quando ocorre a transição do estado

dormindo para o estado *preparando*, se *T* possui subtarefas, alguns testes sobre a consistência dessas subtarefas são feitos, como, por exemplo, um teste sobre a existência de ciclos formados por relacionamentos relacionando tarefas. Assim como o evento de *T*, o evento de cada uma das subtarefas de *T* passa do estado *dormindo* para o estado *preparando*. O *evento de autoria* de *T* passa do estado *preparando* para o estado *preparado* quando todas as subtarefas de *T* forem preparadas e passarem para o estado *preparado*, quando estão prontas para "ocorrer". Se alguma subtarefa falha em passar para o estado *preparado*, o estado do evento de autoria relativo à essa tarefa passa do estado *preparando* para o estado *dormindo*, assim como *T* e todas as subtarefas contidas em *T*.

No estado *preparado*, a tarefa pode ser executada[11] quando chegar o seu momento de execução. Nesse estado, os atores que fazem parte da tarefa não poderão modificá-la. Com *evento de autoria* no estado *preparado*, tarefas podem ser suspensas, representando a impossibilidade temporária de seu início, o que será útil para proibir a sua execução por eventos externos[12] ao modelo de autoria quando, por exemplo, certas alterações em *T* tiverem que ser feitas sem que *T* possa ser iniciada.

Uma tarefa *T* sendo executada está em um estado *ocorrendo*. Se existirem subtarefas em *T*, quando *T* inicia a sua execução, os relacionamentos contidos em *T* serão analisados (condições e ações) para determinar a ordem de realização de suas subtarefas. Uma subtarefa, contida em *T*, só poderá iniciar sua execução se *T* estiver sendo executada.

Depois de serem concluídas, as tarefas têm o estado de seu *evento de autoria* alterado para *concluído*, e não podem mais ser modificadas[13], ou gerar novos documentos. Fica como trabalho futuro, o versionamento de tarefas durante o processo de autoria, para

---

[11] Executar uma tarefa significa cumprir os objetivos propostos para esta tarefa como, por exemplo, criar um determinado documento.

[12] Um evento externo constitui um evento externo à estrutura de relacionamentos definida para a execução do trabalho, como, por exemplo, um início explícito de execução de uma tarefa por um ator.

[13] Modificar uma tarefa significa modificar qualquer atributo da tarefa.

que tarefas já concluídas possam gerar versões, que irão possibilitar a alteração do trabalho dentro do mesmo contexto de tarefas original.

Tarefas, sendo executadas, podem ser suspensas (evento no estado *suspenso*), quando têm sua execução suspensa temporariamente, ou abortadas (evento no estado *abortado*) quando têm sua execução interrompida abruptamente, retornando ao seu estado inicial (com *evento de autoria* no estado *preparado)*. A suspensão de uma tarefa *T* gera a suspensão das subtarefas contidas em *T*. De maneira geral, abortar uma tarefa *T* também faz com que todas as subtarefas de *T* sejam abortadas, exceto quando uma tarefa já foi concluída. Isto porque, quando o *evento de autoria* de uma tarefa *T* passa para o estado *abortado* e em seguida para o estado *preparado*, todas as alterações realizadas em *T* são desfeitas, o que inclui a destruição dos documentos criados, contidos em *T*. Como uma tarefa concluída não pode mais ser modificada, tarefas concluídas não poderão ser abortadas.

Os estados apresentados anteriormente resumem essencialmente o processo de execução de tarefas. No entanto, foi necessário um refinamento para que fosse especificada a máquina de estados completa do *evento de autoria*, ilustrada na Figura 2.3.

Foram definidos novos estados para:

- representar os estados de tarefas compostas por subtarefas, durante etapas do trabalho;
- restringir a alteração para certos estados do *evento de autoria*, dependendo do estado do trabalho realizado em um determinada tarefa;
- para facilitar a gerência do processo de conclusão de tarefas.

Sobre a Figura 2.3, deve-se observar que as elipses tracejadas representam uma agregação de estados com a mesma semântica. É o caso dos estados, que representam a ocorrência de uma tarefa, envolvidos pela elipse com a legenda *Ocorrendo*. Estados, representando a suspensão e a espera de conclusão de uma tarefa, foram omitidos e representados como uma elipse tracejada, para que a máquina de estados ficasse mais clara e objetiva. Esses

estados são mostrados na Figura 2.4, e foram criados para representar as possíveis transições para os estados suspenso, obsoleto e esperando conclusão.

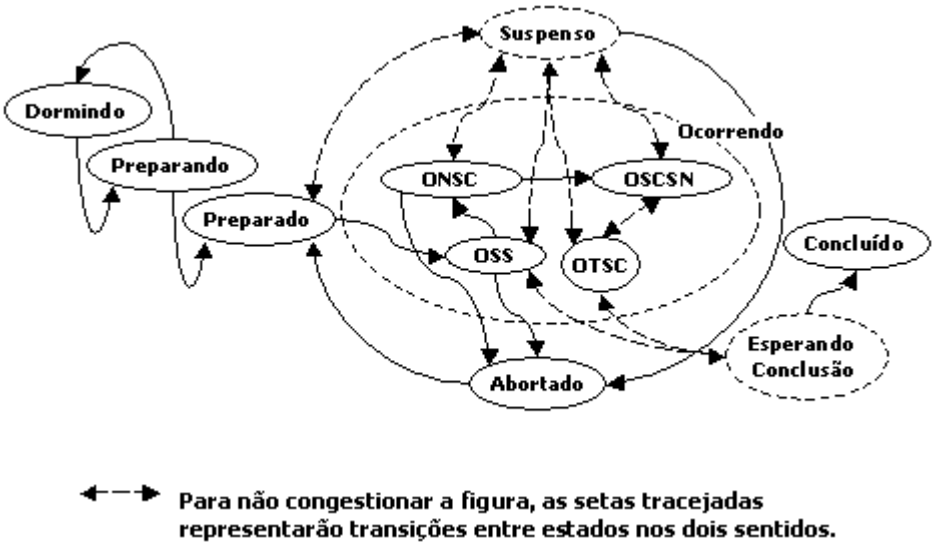


Figura 2.3: Máquina de estados completa do *evento de autoria.*

Uma tarefa *T*, sendo executada, pode ter seu *evento de autoria* nos seguintes estados:

- OSS (*ocorrendo sem subtarefas*): representa a execução de *T*, quando *T* não possui subtarefas;
- ONSC (*ocorrendo com nenhuma subtarefa concluída*): representa a execução de *T*, quando *T* possui uma ou mais subtarefas, mas, ainda, nenhuma foi concluída;
- OTSC (*ocorrendo com todas as subtarefas concluídas*): representa a execução de *T* quando todas as subtarefas já foram concluídas;
- OSCSN (*ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*): representa a execução de *T*, quando o *evento de autoria* de alguma subtarefa está no estado *concluído*, mas não o de todas, ou quando o *evento de autoria* de alguma subtarefa está no estado *ocorrendo com todas as subtarefas concluídas* ou *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*;

Uma tarefa *T*, quando não possui subtarefas a serem realizadas, passa para o estado *ocorrendo sem subtarefas*, quando iniciada. Quando *T* possui subtarefas a serem realizadas, *T* passa para o estado *ocorrendo com nenhuma subtarefa concluída*, quando

é iniciada. A transição, criada na máquina de estados, entre o estado *ocorrendo sem subtarefas* e o estado *ocorrendo com nenhuma subtarefa concluída* ocorre quando subtarefas são inseridas em *T* no estado *ocorrendo sem subtarefas*.

Quando pelo menos uma subtarefa contida em uma tarefa *T* é concluída, o *evento de autoria* de *T* passa para o estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* (quando T possui mais de uma subtarefa), e quando todas o forem, o estado desse evento é alterado para *ocorrendo com todas as subtarefas concluídas*.

Em todos esses estados, o *evento de autoria* pode ser suspenso, mas a transição para outros estados, como o estado *concluído* ou *abortado* depende de em qual estado o *evento de autoria* se encontra.

Apenas tarefas que não possuem subtarefas ou que possuem todas as subtarefas já concluídas, e por isso têm o *evento de autoria* no estado *ocorrendo sem subtarefas* ou *ocorrendo com todas as subtarefas concluídas* podem ser concluídas.

Como tarefas concluídas não podem ser abortadas, tarefas com *evento de autoria* nos estados *ocorrendo com todas as subtarefas concluídas* e *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* não poderão ser abortadas. Da mesma maneira, uma tarefa suspensa poderá ser abortada se isto significar a suspensão de tarefas nos estados *ocorrendo com nenhuma subtarefa concluída* ou *ocorrendo sem subtarefas*.

O estado *esperando conclusão* foi criado para possibilitar revisões do trabalho realizado em uma tarefa *T* por seus atores. Nesse estado, a tarefa não poderá ser alterada[14].

Como foi dito anteriormente, para alterar o estado do *evento de autoria* de uma tarefa é necessária uma permissão especial que deve ser atribuída a um ator. O

[14] Nenhum atributo da tarefa poderá ser modificado.

coordenador responsável por *T* (ator definido com permissão para alterar estados de tarefa) pode concluir a tarefa, se esta estiver sido corretamente realizada e seus objetivos tiverem sido concluídos, ou requisitar alterações, descrevendo as carências da tarefa ou algum comentário importante. No primeiro caso, o evento de *T* passa do estado *esperando conclusão* para o estado *concluído* e *T* não pode ser mais modificada. No segundo caso, o evento de *T* passa do estado *esperando conclusão* para seu estado anterior (*ocorrendo sem subtarefas ou ocorrendo com todas as subtarefas concluídas*) para ser novamente modificada, e o processo se repete até que a tarefa seja concluída (quando o evento passa para o estado *concluído*)[15].

Os estados *suspenso* e *esperando conclusão* foram expandidos em estados com uma mesma semântica, para representar as possíveis transições, a partir de estados diferentes, como pode ser visto a seguir. Foram criados estados de suspensão para o estado *preparado*, e para os diferentes estados de execução de tarefas (*ocorrendo sem subtarefas*, *ocorrendo com nenhuma subtarefa concluída*, *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, *ocorrendo com todas as subtarefas concluídas*). Os estados *esperando conclusão sem subtarefas* e *esperando conclusão com subtarefas* foram criados para representar o processo de conclusão de tarefas a partir dos estados *ocorrendo sem subtarefas* e *ocorrendo com todas as subtarefas concluídas*, respectivamente.

[15] Anotações podem ser bastantes úteis durante esse processo de conclusão de tarefa à medida que facilita a comunicação entre os atores. Anotações serão apresentadas na Seção 3.2.

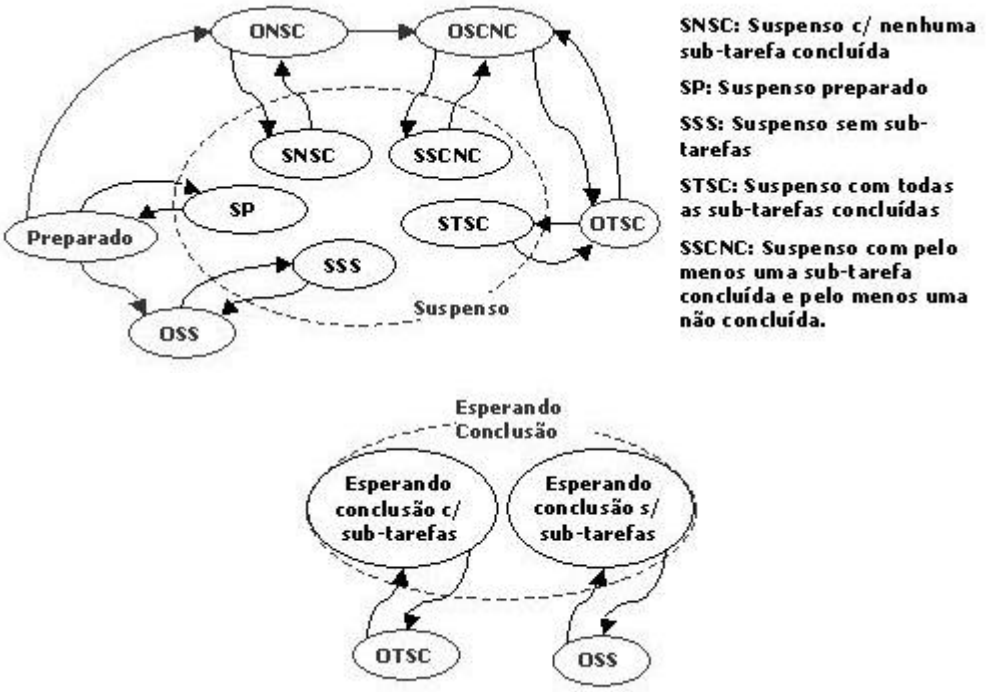


Figura 2.4: Continuação da máquina de estados.

Dependerá desses estados, definidos na máquina de estados do *evento de autoria*, a definição de relacionamentos de causalidade e restrição entre tarefas, que serão mais detalhadamente apresentados na próxima seção.

2.2.3 RELACIONAMENTOS ENTRE TAREFAS

Um *relacionamento* define basicamente três atributos: o *conjunto de pontos terminais de origem*, o *conjunto de pontos terminais de destino* e um atributo *relação*. O conjunto de pontos terminais de origem e destino definem *eventos de autoria* em tarefas, e o atributo relação vai definir relações entre eventos de tarefas, que poderão ser relações de causa ou de restrição.

RELACIONAMENTOS CAUSAIS

No caso de relacionamento causal, o atributo *relação* possui uma operação composta por uma condição e uma ação. Cada condição satisfeita implica no disparo da ação a ela associada. Ações da relação são operações que devem ser executadas nos

pontos terminais de destino dos relacionamentos causais. Condições dizem respeito aos pontos terminais de origem.

As condições de uma relação avaliam valores booleanos e podem ser simples ou compostas. Toda condição simples é expressa por duas condições unárias: uma condição prévia, a ser satisfeita imediatamente antes do instante de tempo em que a condição é avaliada, e uma condição corrente, a ser satisfeita no instante de tempo em que a condição é avaliada. Uma condição simples é satisfeita se tanto a condição prévia quanto a condição corrente são satisfeitas. Tanto a condição prévia quanto a corrente podem receber o valor *VERDADE*, caso elas não sejam relevantes na avaliação da condição simples associada. Os operadores de comparação usados na avaliação das condições simples são:

- = comparação de igualdade
- ≠ comparação de desigualdade

As comparações devem ser realizadas com respeito ao estado do *evento de autoria* de uma tarefa.

Um exemplo de condição a ser satisfeita poderia ser a transição do estado de um *evento de autoria E1*, relativo a uma determinada tarefa, do estado *esperando conclusão* para o estado *concluído*. Nesse caso, a condição prévia receberia o valor (*E1 = esperando conclusão*), e a condição corrente, o valor (*E1 = concluído*). Se o evento de autoria, previamente no estado *esperando conclusão,* passar para o estado *concluído* , a condição simples será satisfeita.

Qualquer expressão de condições baseada nos operadores lógicos $\wedge$ (e), $\vee$ (ou) e $\neg$ (negação) definem uma condição composta.

Condições compostas permitem, por exemplo, a definição de regras para o início de uma tarefa quando mais de uma condição simples for satisfeita, como, quando os estados dos eventos de autoria relativos a mais de uma tarefa forem alterados para o estado *concluído*.

Além disso, a qualquer condição, simples ou composta, poderia ser aplicado um operador simbolizando retardo. O operador retardo aplicado a uma condição *C* é definido da forma $C \circledR [t_1, t_2]$, onde $t_1, t_2 \in \Re$ e $0 \leq t_1 \leq t_2$. Dado que uma condição *C* é verdade num instante *t*, uma condição *C'*, definida como $C \circledR [t_1, t_2]$, é verdade no intervalo de tempo $[t+t_1, t+t_2]$.

Como condições, as ações de um relacionamento causal podem ser simples ou compostas. Estas ações estão diretamente relacionadas a tarefas, mais precisamente a *eventos de autoria* relativos a tarefas.

As ações simples podem ser:

⇒ *Prepara(E)*: se o *evento de autoria E*, relativo a uma tarefa *T*, estiver no estado *dormindo*, ele passa para o estado *preparando*, caso contrário, nenhuma transição ocorre.

⇒ *Inicia(E)*: estando o *evento de autoria E*, próprio à tarefa *T*, no estado *preparado*, nos estados *ocorrendo*[16] ou nos estados *suspenso*, a tarefa passa a ocorrer. No estado *preparado*, o evento de autoria passa para o estado *ocorrendo sem subtarefas* ou *ocorrendo com nenhuma subtarefa concluída*, dependendo se *T* contém subtarefas a serem realizadas ou não. Se o estado inicial for o próprio estado *ocorrendo*, nada acontece. No estado *suspenso preparado* o *evento de autoria* passa para o estado *preparado*, no estado *suspenso sem subtarefas*, passa para o estado *ocorrendo sem subtarefas*, no estado *suspenso com nenhuma subtarefa concluída*, passa para o estado *ocorrendo com nenhuma subtarefa concluída*, no estado *suspenso com pelo menos uma subtarefa concluída*, passa para o estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* e, por último, do estado

[16] Os termos *ocorrendo* e *suspenso*, quando utilizados sozinhos, irão representar os estados relativos a ocorrência (*ocorrendo sem subtarefas*, *ocorrendo com nenhuma subtarefa concluída*, *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, *ocorrendo com todas as subtarefas concluídas*) e suspensão (*suspenso preparado*, *suspenso sem subtarefas*, *suspenso com nenhuma subtarefa concluída*, *suspenso com pelo menos uma subtarefa concluída e pelo menos uma não concluída* ou *suspenso com todas as subtarefas concluídas*) de uma tarefa.

suspenso com todas as subtarefas concluídas, passa para o estado *ocorrendo com todas as subtarefas concluídas*.

⇒ *Termina(E)*: altera o estado do *evento de autoria E* de *ocorrendo sem subtarefas* ou *ocorrendo com todas as subtarefas concluídas* para *esperando conclusão sem subtarefas* ou *esperando conclusão com subtarefas* respectivamente. Se o estado inicial for qualquer outro, nada acontece.

⇒ *Suspende(E)*: altera o estado do *evento de autoria E* de *ocorrendo* para *suspenso*. Se o estado inicial for qualquer outro, nada acontece.

⇒ *Reassume(E)*: altera o estado do *evento de autoria E* de *suspenso* para *ocorrendo*, retomando a execução do mesmo ponto antes da execução do *evento de autoria* ser interrompida, e do estado *esperando conclusão* para *ocorrendo*. Se o estado inicial for qualquer outro, nada acontece.

⇒ *Conclui(E)*: altera o estado do *evento de autoria E* de *esperando conclusão* para o estado *concluído*. Se o estado inicial for qualquer outro, nada acontece.

⇒ *Indica Conclusão(E)*: altera o estado do *evento de autoria E* de *ocorrendo com nenhuma subtarefa concluída*, para *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, ou para *ocorrendo com todas as subtarefas realizadas*, ou do estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* para o estado *ocorrendo com todas as subtarefas realizadas*. A alteração dependerá de cada tarefa e do estado de suas subtarefas. Se o estado inicial for qualquer outro, nada acontece.

⇒ *Aborta(E)*: altera o estado do *evento de autoria E* de *ocorrendo* ou *suspenso* para *abortado*, e imediatamente depois para *preparado*. Se o estado inicial for qualquer outro, nada acontece.

Uma ação composta é formada por uma expressão de ações baseada nos operadores | (paralelo) e → (seqüencial), definindo a ordem de execução de cada elemento da ação. Será possível, através da definição de ações compostas, a definição do início de tarefas em paralelo ou de maneira seqüencial. No primeiro caso, não é definida uma seqüência para o início das tarefas, apenas que tarefas deverão ser iniciadas aproximadamente no mesmo

momento. Para que duas tarefas *T1* e *T2*, com eventos *e1* e *e2* associados, sejam iniciadas em paralelo pode ser definida a seguinte ação: Inicia (e1) | Inicia (e2).

É importante notar que determinadas ações só serão relevantes para determinados estados do evento de autoria, por isso deve-se ter cuidado na definição de relacionamentos entre tarefas, para que as ações definidas em um determinado relacionamento sejam compatíveis com os estados do evento de autoria relativo a cada tarefa que participa desse relacionamento. Caso contrário, pode ser que o trabalho realizado em tarefas não seja corretamente realizado.

RELACIONAMENTOS DE RESTRIÇÃO

Um *relacionamento de restrição* define um conjunto de pontos terminais de destino nulo. O conjunto de pontos terminais de origem define *eventos de autoria* relativos a tarefas, e o atributo *relação* vai definir restrições entre esses eventos.

A semântica da operação é que todas as condições prévias, definidas para o evento de autoria de uma determinada tarefa, que forem *VERDADE* em um dado instante de tempo devem ter nas condições correntes correspondentes o mesmo valor (*VERDADE* ou *FALSO*), nesse mesmo instante de tempo.

Relacionamentos de restrição poderão definir, por exemplo, a conclusão de duas tarefas ao mesmo tempo, ou o início de uma tarefa apenas após a conclusão de uma outra tarefa[17]. Na primeira situação, o relacionamento entre duas tarefas poderia ser definido da seguinte maneira: Uma condição para uma tarefa *T1* definiria como condição prévia o estado do evento de autoria *E1*, relativo a *T1*, no estado *esperando conclusão sem subtarefas* (*E1 = esperando conclusão sem subtarefas*), e a condição corrente (quando a tarefa fosse concluída), no estado concluído (*E1 = concluído*). Outra condição para uma tarefa *T2* definiria como condição prévia o estado do evento de autoria *E2*, relativo a *T2*,

[17] Não existe aqui, um relacionamento de causalidade entre as tarefas. Não quer dizer que uma tarefa deva ser iniciada após a conclusão de outra tarefa, apenas que uma tarefas só pode ser iniciada, quando uma outra já estiver concluída.

no estado *esperando conclusão sem subtarefas* (*E2 = esperando conclusão sem subtarefas*), e a condição corrente (quando a tarefa fosse concluída), no estado concluído (*E2 = concluído*). A condição composta do elo será satisfeita se as condições prévias das condições simples forem VERDADE e as condições correntes também forem VERDADE em um dado instantes.

2.2.4 UM EXEMPLO DE EXECUÇÃO DE TAREFAS

A Figura 2.5 representa a estrutura de uma tarefa *T1*. Esta estrutura ilustra a ordem de execução de *T1* de acordo com os relacionamentos definidos.

A estrutura define:

- Uma tarefa *T1*, representando o trabalho como um todo, com subtarefas *T1.1* e *T1.2*;
- *T1.1* com subtarefas *T1.1.1* e *T1.1.2*;
- Relacionamentos causais (R) em *T1* que definem:
  - o início de *T1.1* quando *T1* é iniciada (R1);
  - a alteração do estado do *evento de autoria* de *T1*, quando o estado do evento de *T1.1* for alterado para *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* (R2);
  - o início de *T1.2* quando *T1.1* é concluída (R3);
  - a alteração do estado do *evento de autoria* de *T1* para o estado *ocorrendo com todas as subtarefas concluídas* quando *T1.2* for concluída (R4);
- Relacionamentos em *T1.1* que definem o início de *T1.1.1* e *T1.1.2* em paralelo (R5), a alteração do estado de *T1.1* quando uma das subtarefas é concluída (R6), e a sua conclusão (indicação de conclusão de suas subtarefas) quando todas as subtarefas forem concluídas (R7).

  Como exemplo, o atributo *relação* do relacionamento R7 poderia ser definido da seguinte forma: Condição: <VERDADE, e(T1.1.1) = *concluído*> $\vee$ <VERDADE, e(T1.1.2) = *concluído*> $\wedge$ <VERDADE, e(T1.1) = *ocorrendo com todas as*

subtarefas concluídas> Ação: Termina(e(T1.1)), onde e(X) representa o evento de autoria relativo a uma tarefa X.

T1 será criada no estado *dormindo*, assim como as suas subtarefas e relacionamentos que definem a sua execução. A tarefa poderá ser executada depois de passar pelos estados *preparando* e *preparado*, assim como todas as suas subtarefas.

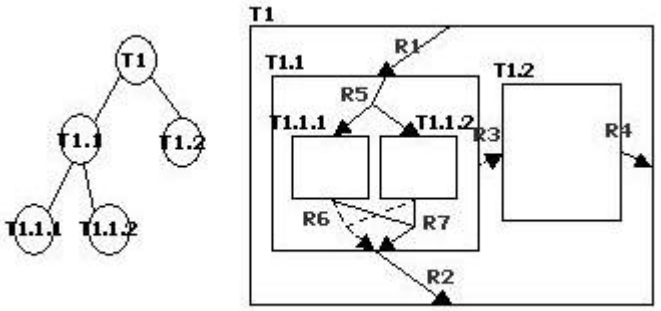


Figura 2.5: Execução de tarefas.

Para ser efetivamente executada, quando chegar a hora da execução, o *evento de autoria* de *T1* deve passar do estado *preparado* para o estado *ocorrendo com nenhuma subtarefa concluída*, já que existem subtarefas a serem executadas (*T1.1* e *T1.2*).

Quando essa transição do estado do evento de *T1* acontece (condição de R1), os relacionamentos contidos em *T1* são analisados e o *evento de autoria* de *T1.1* tem seu estado alterado de *preparado* para *ocorrendo com nenhuma subtarefa concluída* (ação de R1), quando sua execução é iniciada.

Quando ocorre a transição do estado do evento de *T1.1* de *preparado* para *ocorrendo com nenhuma subtarefa concluída*, um relacionamento em *T1.1* (R5), que tem como condição esta transição, inicia *T1.1.1* e *T1.1.2* em paralelo (ação de R5) que têm os estados de seus eventos alterados para *ocorrendo sem subtarefas*, já que não possuem subtarefas a serem realizadas.

Depois que as tarefas *T1.1.1* e *T1.1.2* são executadas por completo, o evento relativo a cada uma delas deve passar do estado *ocorrendo sem subtarefas* para o estado

concluído (pode haver algum debate sobre a conclusão da tarefa, com várias transições para o estado *esperando conclusão*).

Quando ocorre a primeira conclusão, o *evento de autoria* de *T1.1* passa do estado *ocorrendo com nenhuma subtarefa concluída* para o estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* (ação de R6). Esta transição no estado do *evento de autoria* de *T1.1* gera a transição do estado do *evento de autoria* de *T1* de *ocorrendo com nenhuma subtarefa concluída* para o estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* (ação de R2). Quando as duas subtarefas são concluídas (condição de R7), o *evento de autoria* de *T1.1* passa para o estado *ocorrendo com todas as subtarefas concluídas* (ação de R5). O que tiver que ser feito em *T1.1* é feito até que a tarefa seja concluída e o seu evento passe para o estado *concluído*.

Quando o *evento de autoria* de *T1.1* passa para o estado *concluído*, *T1* passa para o estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* (nesse caso, já está, pela ação de R2). O relacionamento em *T1* (R3) que possui como condição a transição do estado do *evento de autoria* de *T1.1* para o estado *concluído* (condição de R3) é analisado e *T1.2* é iniciada (ação de R3). Nesse ponto, o *evento de autoria* de *T1.2* passa do estado *preparado* para o estado *ocorrendo sem subtarefas*, até que a tarefa seja concluída.

Quando o *evento de autoria* de *T1.2* passa para o estado *concluído* (condição de R4), *T1* passa do estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* para o estado *ocorrendo com todas as subtarefas concluídas* (ação de R4), já que não possui mais subtarefas a serem realizadas.

O trabalho como um todo é finalizado quando *T1* é concluída (com *evento de autoria* no estado *concluído*).

Durante o processo de execução de tarefas, alterações poderão ser realizadas e tarefas poderão ser adicionadas ou removidas. Essa autoria, todavia, deve ser realizada

com cautela para que a execução do trabalho permaneça consistente após as alterações. A próxima seção irá apresentar algumas observações e restrições a autoria de tarefas.

2.2.5 ALTERAÇÕES EM TAREFAS

Nesta seção, serão apresentadas algumas observações sobre a autoria de tarefas em diferentes fases do processo de trabalho.

Quando se deseja alterar uma tarefa *T*, incluindo ou removendo subtarefas e relacionamentos, deve-se levar em conta o estado do *evento de autoria* associado às tarefas já existentes que possuem relação com a subtarefa que será removida, ou que devem ter algum tipo de relacionamento com a subtarefa que será inserida. Fica a cargo dos atores responsáveis por estas alterações, a preocupação com a manutenção e corretude do trabalho.

De maneira geral, subtarefas contidas em *T* que irão possuir algum relacionamento com novas subtarefas a serem inseridas no *conjunto de subtarefas* de *T*, ou que possuem relacionamentos com subtarefas a serem removidas de *T* devem ser suspensas até que a operação seja realizada. Esse procedimento deverá suspender trabalhos relacionados, evitando que ocorra alguma inconsistência durante a alteração em *T*. Esta alteração usualmente deve ser seguida ou antecedida por alterações em relacionamentos contidos em *T*.

É importante observar que relacionamentos podem se tornar inconsistentes, a partir da remoção de uma subtarefa de *T*, quando uma relação pode passar a conter uma referência nula. Nesse caso, a subtarefa, não removida, referida nesta relação pode não mais ser referenciada, e nunca ser iniciada, por exemplo. Em muitos casos, relacionamentos deverão ser removidos antes que a subtarefa o seja, para evitar que sejam executados de forma inconsistente. Embora não sejam feitas restrições sobre a remoção de tarefas, torna-se fundamental essa preocupação (responsabilidade do ator que tiver permissão para realizar tal operação).

Aqui, algumas observações sobre a autoria de tarefas devem ser apresentadas. Estas observações dirão respeito à inserção de uma tarefa *T1* em uma tarefa *T*, ou à remoção de outra tarefa *T2* contida em *T*.

⇒ Como dito anteriormente, o ator responsável pela inserção de *T1* em *T*, ou a remoção de *T2* de *T*, deve suspender as subtarefas contidas em *T* que irão ter ou têm algum relacionamento com a subtarefa que será inserida ou excluída, quando julgar necessário. A suspensão das subtarefas relacionadas irá evitar possíveis inconscistências no trabalho ocorram, assim como evitar que certas ações, ocasionadas por condições de relacionamentos se percam. Ainda, o ator será responsável pela manutenção e corretude dos novos relacionamentos criados ou dos relacionamentos já existentes, contidos em *T*.

⇒ Os atores, responsáveis pela autoria em *T*, devem se preocupar em retornar as tarefas que foram suspensas a seu estado anterior depois da alteração.

⇒ Uma tarefa *T1*, que deve ser inserida em *T*, inicialmente é criada em *T* com *evento de autoria* no estado *dormindo*, e deve passar pelos estado *preparando e preparado* antes de ser iniciada a sua execução.

⇒ Quando se deseja incluir *T1* em *T*, com *evento de autoria* no estado *ocorrendo sem subtarefas*, o estado do evento de *T* deve ser alterado para o estado *ocorrendo com nenhuma subtarefa concluída*, já que agora *T* irá possuir uma subtarefa que deve ser realizada.

⇒ Quando se deseja incluir *T1* em *T*, com *evento de autoria* no estado *ocorrendo com todas as subtarefas concluídas*, o estado do evento de *T* deve ser alterado para o estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, já que mais uma subtarefa terá que ser realizada.

⇒ Quando se deseja remover *T2* contida em *T*, serão removidas também as possíveis subtarefas de *T2*.

⇒ Quando se deseja remover *T2* de *T*, se *T2* é a única subtarefa contida em *T*, o *evento de autoria* de *T* deve ser alterado para o estado *ocorrendo sem subtarefas*, caso contrário devem ser analisados os possíveis estados de *T2* e *T*:

- Se o *evento de autoria* de *T* está no estado *ocorrendo com nenhuma subtarefa concluída*, ou no estado *ocorrendo com todas as subtarefas concluídas*, o estado do evento de autoria não é alterado;
- Se o *evento de autoria* de *T* está no estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, e se o *evento de autoria* de *T2* está no estado *concluído*, *ocorrendo com todas as subtarefas concluídas*, ou *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, e não existem mais subtarefas de *T* em um desses estados, o evento de autoria de *T* deve passar para o estado *ocorrendo com nenhuma subtarefa concluída*, caso contrário, o *evento de autoria* de *T* não tem o seu estado alterado;
- Se o *evento de autoria* de *T* está no estado *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída*, e se o *evento de autoria* de *T2* está no estado *ocorrendo sem subtarefas* ou *ocorrendo com nenhuma subtarefa concluída*, e não existem mais subtarefas de *T* em um desses estados, o *evento de autoria* de *T* deve passar para o estado *ocorrendo com todas as subtarefas concluídas*, caso contrário, seu estado não é alterado.

⇒ Quando se deseja inserir *T1* em *T*, ou remover *T2* de *T*, no estado *preparado*, deve-se considerar o início da execução de *T* durante a alteração e, se for necessária à sua suspensão. Deve-se notar que a suspensão de *T*, nesse caso, poderá gerar a suspensão de outras tarefa relacionadas à *T*, ou a outras tarefas suspensas relacionadas à *T*, para evitar que ações geradas por transições em estados de *evento de autoria* se percam enquanto a alteração é realizada.

⇒ Relacionamentos, assim como tarefas, poderão ser inseridos em *T* (sem que formem ciclos entre subtarefas de *T*) ou removidos de *T*, por um ator com direito de acesso adequado.

⇒ A única restrição que será feita para a criação de relacionamentos é a seguinte: Uma subtarefa *T1*, com *evento de autoria* no estado *dormindo*, contida em *T*, só poderá ter relacionamentos com outras subtarefas contidas em *T* com *evento de autoria*, também no estado *dormindo*, ou com evento de autoria em um estado suspenso, que

só poderá voltar ao seu estado inicial depois que o evento de autoria de *T1* for alterado para o estado *preparado*. Essas restrições irão garantir o início correto do trabalho em *T1*, que só deve ser ter sua execução iniciada depois de ser consistentemente criada e preparada.

⇒ Nesse processo de *autoria de tarefas*, deve-se levar em conta, ainda, o fato de poder haver mais de uma modificação na mesma tarefa ao mesmo tempo. Esse problema deve ser resolvido por um controle de concorrência que evite a realização de alterações conflitantes, tanto em documentos durante o trabalho cooperativo, como na autoria de tarefas.

Deve-se observar, que alterações realizadas em uma tarefa já ocorrendo devem ser realizadas de maneira muito consciente, já que toda a estrutura de execução do trabalho poderá ser alterada. A possibilidade de alteração nessa estrutura de tarefas deve ser estabelecida através de definições de permissão de acesso para cada tarefa, possibilitadas por mecanismos de coordenação necessários à cooperação, que serão apresentados na Seção 3.3.

O *modelo de tarefas*, apresentado neste capítulo, servirá de base para a elaboração de um *modelo de cooperação* que irá oferecer mecanismos de colaboração, comunicação e coordenação, essenciais para o trabalho cooperativo.

Para a especificação dos mecanismos de cooperação, que serão apresentados no próximo capítulo e que deverá proporcionar a cooperação assíncrona, síncrona e semi-síncrona, espera-se que estejam dispostos certos mecanismos que não serão abordados neste trabalho:

- um controle de notificação, que dará suporte à consciência de grupo entre os participantes do processo e à propagação do trabalho entre os atores (terá forte influência na comunicação entre atores de tarefas); e,
- um controle de versão, para possibilitar à autoria cooperativa semi-síncrona.

3. COOPERAÇÃO NO MODELO DE TAREFAS

Neste capítulo, será apresentado como a cooperação poderá ser realizada utilizando-se o modelo de tarefas como suporte. Depois de serem analisadas os meios que irão possibilitar a colaboração (Seção 3.1), a comunicação (Seção 3.2) e a coordenação (Seção 3.3), essenciais ao processo cooperativo, será apresentado o *modelo de cooperação* (Seção 3.4), que dará suporte à autoria cooperativa.

3.1 COLABORAÇÃO

A colaboração entre atores e a troca de informação durante a autoria de documentos será realizada segundo a estrutura de tarefas definida para a realização do trabalho. Em tarefas, os documentos utilizados durante o trabalho serão criados e alterados.

3.1.1 VISIBILIDADE DE DOCUMENTOS EM TAREFAS

Atores, durante a autoria cooperativa de documentos, poderão trabalhar em ambientes particulares ou públicos, dependendo da restrição que se queira fazer a respeito da visibilidade de um trabalho que deve ser realizado. De maneira geral, a noção de ambiente particular de trabalho servirá para restringir a visibilidade de um trabalho a um conjunto de atores, quando esse último faz parte de um outro trabalho maior, enquanto ambientes públicos irão proporcionar a troca de informações entre ambientes particulares.

Tarefas irão representar ambientes de trabalho, e dependendo de como documentos serão dispostos em tarefas, esses poderão ser públicos ou privados a determinados atores e tarefas. A idéia de ambientes de trabalho públicos e privados será útil na definição da estrutura de tarefas que irá compor o trabalho cooperativo e servirá para introduzir a idéia de visibilidade de documentos.

Uma tarefa *T* representa um ambiente público de trabalho para os atores pertencentes a *T* e às subtarefas recursivamente contidas em *T*, ao mesmo tempo que

representa, um ambiente particular de trabalho em relação a todas as demais tarefas. Contidos em *T*, documentos serão *visíveis* à *T* e às subtarefas recursivamente contidas em *T*.

Mais precisamente, diz-se que um documento é *visível* a uma tarefa *T* e a atores de *T*, quando está contido em *T*, ou em uma tarefa na qual *T* está recursivamente contida.

A Figura 3.1 apresenta um exemplo simples, que ilustra a definição de tarefas de maneira a proporcionar ambientes de trabalhos particulares e públicos, de acordo com o trabalho que deve ser realizado por atores em cooperação. Esse trabalho será representado pela tarefa *T1*.

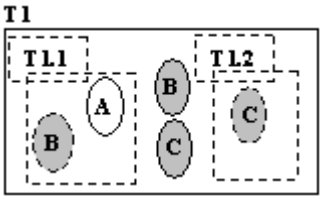


Figura 3.1: Ambiente de colaboração.

A tarefa *T1* representa o ambiente público de trabalho para dois atores, responsáveis pelo trabalho, e contém os documentos *B* e *C*, que devem ser compartilhados em *T1*. Foram definidas as subtarefas *T1.1* e *T1.2* para representar um ambiente particular a cada ator, onde o trabalho particular será realizado.

Nesse exemplo, o documento *A*, contido em *T1.1*, será visível apenas ao ator de *T1.1*, enquanto os documentos *B* e *C* serão visíveis às duas subtarefas.

Para que se torne visível a determinadas tarefas, o trabalho realizado em uma subtarefa *ST*[18], recursivamente contida em *T*, poderá ser exposto em *T* ou em uma outra tarefa *T'*, recursivamente contida em *T*, na qual *ST* está recursivamente contida, se for permitido pelo controle de acesso. No primeiro caso, o trabalho seria visível a *T* e às subtarefas recursivamente contidas em *T*, enquanto no segundo caso, seria visível a *T'* e às subtarefas recursivamente contidas em *T'*. Esse recurso será muito importante durante a

[18] Trabalho esse particular a *ST* e às subtarefas recursivamente contidas em *ST*.

autoria cooperativa de documentos em tarefas, possibilitando que um trabalho particular possa ser exposto em um ambiente público, e continuado em outras tarefas.

Deve-se notar ainda que, mesmo visível a certos atores e tarefas, pode ser necessária a definição de regras de acesso a documentos. Esse tipo de restrição depende dos mecanismos de coordenação definidos para cada tarefa durante o trabalho cooperativo, mais precisamente o que diz respeito a mecanismos de controle de acesso.

Dessa maneira, tarefas deverão ser definidas, de forma a criar ambientes de trabalho particulares e públicos que forneçam a privacidade e visibilidade necessária a um conjunto de atores, sobre determinados documentos. A autoria cooperativa de documentos dependerá dessa estrutura definida para a realização do trabalho.

3.1.2 AUTORIA DE DOCUMENTOS EM TAREFAS

Os documentos utilizados durante a autoria cooperativa poderão ser versões de outros documentos ou não, dependendo se o sistema de autoria irá suportar ou não o uso de versões. Aos documentos que geram versões dar-se-á o nome de documentos versionáveis[19], e aos que não geram versões, de documentos não versionáveis.

Basicamente, quando se deseja realizar uma alteração em um documento versionável *V*, em um estado consistente, uma versão *V'* deve ser criada a partir de *V*, onde as alterações poderão ser realizadas. Dependerá de cada sistema em particular, a forma de implementar mecanismos de suporte à versão. Já uma alteração em um documento não versionável, poderá ser feita diretamente no documento de acordo com as regras de acesso estabelecidas para isso.

Atores engajados em um processo colaborativo, utilizando documentos não versionáveis, devem ser mais responsáveis em relação às alterações realizadas nesses documentos, já que alterar um documento *D* irá significar, diretamente, a propagação de alterações em todos os documentos que tenham alguma relação (por inclusão, ou através de

[19] Documentos versionáveis constituem a base da autoria cooperativa semi-síncrona.

elos) com *D*. O uso de versões possibilitará uma maior consistência do trabalho ao longo do processo de autoria à medida que o trabalho poderá ser realizado sobre versões de dados consistentes. Por outro lado, o trabalho em versões pode tornar a colaboração mais complexa, à medida que muitas vezes pode ser necessário que alterações realizadas em diferentes versões de um mesmo documento sejam refletidas em uma única versão. Nesse caso, mecanismos que possibilitem fusões de versões de forma automática podem ser bastante interessantes. Esses mecanismos não serão tratados neste trabalho, cabendo aos atores a preocupação com a organização de alterações em versões de um mesmo dado.

AUTORIA DE DOCUMENTOS EM TAREFAS

Durante a autoria de documentos em uma tarefa, alguns pontos devem ser observados, assim como algumas restrições devem ser feitas.

A primeira observação que deve ser feita é que a autoria cooperativa de documentos poderá ser realizada, em uma tarefa *T*, sobre documentos contidos em *T* ou contidos em uma tarefa na qual *T* está recursivamente contida (documentos visíveis à *T*, conforme apresentado na Seção 3.1.1). Nesse último caso, esses documentos farão parte de mais um atributo de tarefa, o ***conjunto de documentos contidos por recursão*** (*Cdcr*).

Diz-se que um documento *D* é um *componente* de *Cdcr*, e que *D* está *contido por recursão* em *T*, se está contido em uma tarefa na qual *T* está recursivamente contida. Um documento *D*, contido por recursão em uma tarefa, pode estar contido por recursão em outras tarefas recursivamente contidas em uma tarefa *T* onde *D* está efetivamente contido.

Cabe aqui ressaltar que, como mencionado em 2.2, um documento *D* só pode estar contido em uma única tarefa.

Esse novo atributo foi criado para diferenciar documentos utilizados em uma tarefa *T* que estão contidos em *T*, daqueles apenas visíveis a *T*, possibilitando a autoria cooperativa de documentos segundo ambientes de trabalho públicos e privados, de acordo com o apresentado na Seção 3.1.1.

O exemplo que se segue irá permitir que várias situações, possíveis de ocorrer durante a autoria cooperativa de documentos, possam ser identificadas ao longo desta seção.

O exemplo apresentado na Figura 3.2 retrata a autoria cooperativa de um documento hipermídia (*Dh*) por um grupo de atores, a partir do seguinte cenário:

⇒ Foram definidas as tarefas *T*, *T1* e *T2*, para formar a estrutura de tarefas para a autoria do documento *Dh*, que, ao final do trabalho realizado em *T*, será composto por duas seções representadas pelos documentos *S1* e *S2*, como pode ser visto na Figura 3.3.

- *Dh* deverá ser criado em *T*, onde foi inicialmente definido contendo uma única seção, composta pelo documento *S2*.
- *S2* será modificado em *T2*, segundo determinados objetivos definidos no momento da criação da tarefa.
- O documento *S1* deverá ser criado na tarefa *T1*, criada exatamente com esse objetivo.

⇒ Em *T* foram definidos os documentos *Dh*, *S2* e *B*, contidos em *T*, conforme a estrutura apresentada na Figura 3.2.

⇒ Em *T1* e *T2* foram definidos os documentos *A* e *S1*, e *C*, contidos em *T1* e *T2* respectivamente. Ainda, estarão *contidos por recursão* os documentos *S2*, em *T2*, e *B*, em *T1* (na Figura 3.2(a)), quando tiverem que ser alterados nessas tarefas.

A partir do cenário que foi apresentado, a autoria do documento *Dh* dependerá se os componentes desse documento serão documentos versionáveis ou não. Para mostrar a diferença entre os dois casos, o documento *B*, na Figura 3.2 (b), contido em *T*, foi definido como um documento versionável, em um estado permanente, que só poderá ser alterado em uma nova versão, enquanto os demais documentos são documentos que poderão ser diretamente alterados. Na Figura 3.2 (a), todos os documentos são documentos não versionáveis e poderão ser diretamente alterados. Deve-se notar que, no primeiro caso, *B'*, uma versão de *B* em *T1*, está contido em *T1*, e fará parte do *conjunto de documentos* de *T1*, enquanto no segundo caso, *B* está *contido por recursão* em *T1*, e fará parte do *conjunto de documentos contidos por recursão* de *T1*, como dito anteriormente.

A Figura 3.2(a) ilustrará o cenário para a autoria de documentos não versionáveis, enquanto a Figura 3.2 (b) ilustrará o cenário para a autoria de documentos, no qual o documento *B* é versionável. Em ambas as figuras, documentos contidos por recursão em uma tarefa são representados por uma elipse com contorno tracejado, enquanto documentos contidos em uma tarefa, por uma elipse com contorno contínuo.

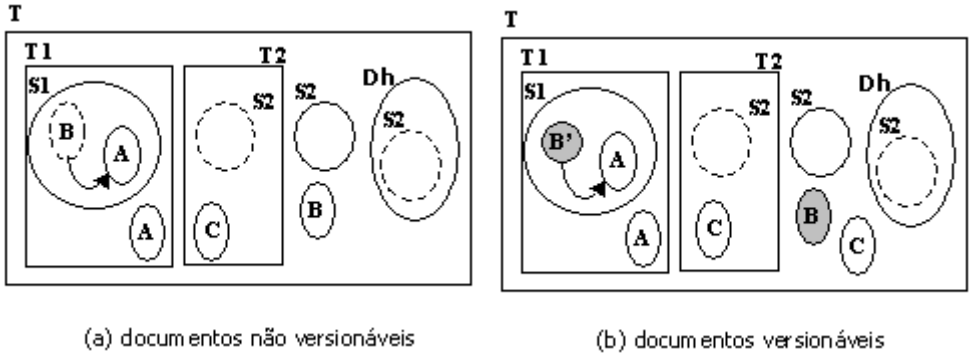


Figura 3.2: Autoria de documentos.

Sobre os cenários de autoria, algumas observações devem ser feitas:

1- Se um documento está contido em *T*, seus componentes devem estar contidos em *T* ou em uma tarefa de um aninhamento de tarefas onde *T* está recursivamente contida (contidos por recursão em *T*).

   No exemplo, para que o documento *S2* seja modificado em *T2* pela inserção de um documento *C*, definido inicialmente em *T2*, esse último deverá obrigatoriamente fazer parte do *conjunto de documentos* de *T*, para que a consistência apresentada no item 1 seja mantida. Isso porque S2 está contido em *T*, e *C*, depois da inserção em *S2*, em *T2*, é um componente de *S2*.

2- Elos contidos em *T* só poderão ser definidos entre documentos contidos em *T* ou contidos por recursão em *T*. Um elo não poderia ser definido, por exemplo, entre *A* e *C*, contidos em *T1* e *T2*, respectivamente.

3- Para que os resultados de um trabalho particular a uma tarefa *T* (documentos contidos em *T*), sejam visíveis a outras tarefas, além de *T* e das tarefas recursivamente contidas em *T*, deve ser possível a transferência desses resultados para uma tarefa *T'*, onde *T*

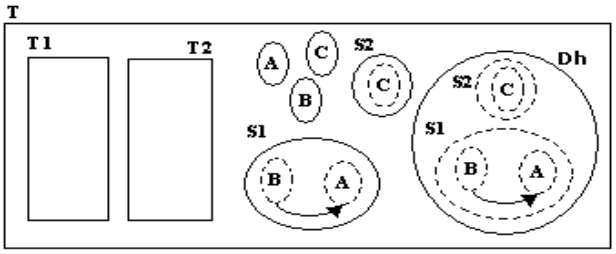
está recursivamente contida, que forneça a visibilidade desejada, de acordo com o que foi definido em 3.1.1. Mais precisamente, a transferência de documentos entre tarefas significará a transferência de documentos entre *conjuntos de documentos* de tarefas[20].

No exemplo, o documento *S1*, contido em *T1*, se tornaria visível a *T* e às subtarefas recursivamente contidas em *T*, quando fosse inserido[21] no *conjunto de documentos* de *T*.

4- Quando um determinado documento *D* é transferido para uma tarefa, todos os documentos contidos em *D* também devem ser transferidos para a mesma tarefa.

Quando o documento *S1*, inicialmente contido em *T1*, for transferido para a tarefa *T*, seus componentes, os documentos *A* e *B, ou B'*, na Figura 3.2(b), também deveriam ser transferidos para a mesma. Como *B*, na Figura 3.2(a) já está contido em *T*, esse não será transferido, ao contrário dos documentos *A* e *B'*.

A figura seguinte ilustra o resultado da autoria do documento *Dh*, exposto em *T*, de acordo com os cenários anteriores.



(a) documentos não versionáveis

[20] A transferência de documentos entre tarefas deve ser regulada por mecanismos de coordenação.

[21] E, automaticamente, removido do conjunto de documentos de *T1*.

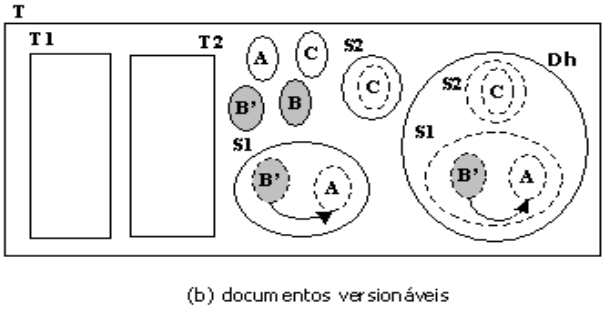


Figura 3.3: Resultado da autoria em *T*.

Durante a autoria cooperativa de documentos, documentos poderão ser inseridos e removidos do *conjunto de documentos* de uma tarefa. Do mesmo jeito que durante a autoria de tarefas, essas inserções e remoções de documentos só deverão ser realizadas por atores com permissão para isso, com bastante cuidado, evitando-se que inconsistências sejam criadas, assim como referências nulas.

Por último, a autoria de documentos, realizada em uma tarefa, irá depender do estado dessa tarefa, ou mais precisamente, do estado do *evento de autoria* relativo à tarefa em questão:

- A autoria de documentos em uma tarefa *T* poderá ocorrer enquanto essa tarefa está sendo executada (quando o *evento de autoria* de *T* está nos estados *ocorrendo sem subtarefas*, *ocorrendo com nenhuma subtarefa concluída*, *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* e *ocorrendo com todas as subtarefas concluídas*). Quando *T* é suspensa, a autoria de documentos também deverá ser suspensa em *T*.
- Quando o estado do *evento de autoria* de uma tarefa *T* é alterado para o estado *concluído*, a autoria de documentos em *T* também é concluída, e o *conjunto de documentos* de *T* não poderá ser mais modificado. Por isso, durante o processo de conclusão da tarefa, quando o *evento de autoria* de *T* se encontra no estado *esperando conclusão*, as últimas alterações em documentos contidos em *T* devem ser sinalizadas para serem realizadas enquanto o *evento de autoria* estiver em um estado ocorrendo. Essas alterações incluem a inserção de documentos contidos em *T* em tarefas adequadas, para refletir o trabalho realizado em *T*.

Durante todo o processo colaborativo, durante a autoria em documentos versionáveis ou não, atores engajados em trabalhos relacionados devem estar cientes de determinadas alterações realizadas em documentos utilizados durante o trabalho. Um controle de notificação deverá atuar nesse sentido, notificando alterações em documentos a atores interessados. Notificações farão parte dos mecanismos de comunicação que serão apresentados na próxima seção.

3.2 COMUNICAÇÃO

A comunicação irá possibilitar a troca de informações durante o processo cooperativo, e representará um fator fundamental para a interação do grupo de trabalho, tanto na realização de um trabalho assíncrono como de um trabalho síncrono ou semi-síncrono.

As regras de visibilidade de documentos, definidas na Seção 3.1.1, fornecem o primeiro passo para viabilizar a comunicação entre os participantes de tarefas à medida que possibilita a troca de informações (documentos) coordenada entre atores. *Anotações* irão possibilitar uma forma de comunicação mais explícita, podendo ser feitas sobre documentos, sobre o trabalho (tarefas) realizado em algum momento, ou mesmo sobre outras anotações. Notificações também irão exercer um papel importante durante a comunicação entre atores engajados em processo cooperativo.

3.2.1 ANOTAÇÕES

Anotações podem ser vistas como documentos, que representam comentários, que podem ou devem ser compartilhadas por atores interessados. Esses comentários podem ser sugestões de trabalho, alterações em documentos, podem representar explicações sobre alterações, ou qualquer outro tipo de informações que possam ser agregadas a documentos, ou partes de documentos ou tarefas, sem que esses sejam alterados.

Além de referenciarem documentos ou tarefas, anotações poderão referenciar anotações, com o objetivo de responder a comentários, ou mesmo a indagações feitas por atores. Anotações, nesse sentido, poderão proporcionar diálogos assíncronos ou síncronos

entre participantes de uma tarefa, devendo permanecer em um ***estado*** ativo até que seus objetivos sejam cumpridos, e tornadas obsoletas quando esses o forem.

Cada tarefa *T* terá mais um atributo denominado ***conjunto de anotações***, que conterá as anotações sobre documentos contidos ou contidos por recursão em *T*, sobre subtarefas contidas em *T* ou sobre a própria tarefa *T*, ou sobre anotações contidas em *T*. Diz-se que uma anotação *A* está contida em uma tarefa *T*, se *A* pertence ao *conjunto de anotações* de *T*. Cada anotação será criada por um determinado ator, representando o ***autor*** da anotação, um atributo de anotação.

Anotações serão relacionadas a tarefas, a documentos ou a anotações através de *elos de anotação*. O ***conjunto de elos de anotação*** *(C_{ea})* constitui mais um atributo de uma tarefa *T*. Diz-se que um elo de anotação é um componente de C_{ea}, e que está contido em *T*, se pertence a C_{ea}.

Anotações terão a mesma visibilidade que aquela definida para documentos em tarefas. Uma anotação *A* será particular ou pública a uma determinada tarefa dependendo da tarefa onde *A* está contida.

O exemplo, apresentado na figura seguinte, ilustra algumas situações onde anotações foram realizadas.

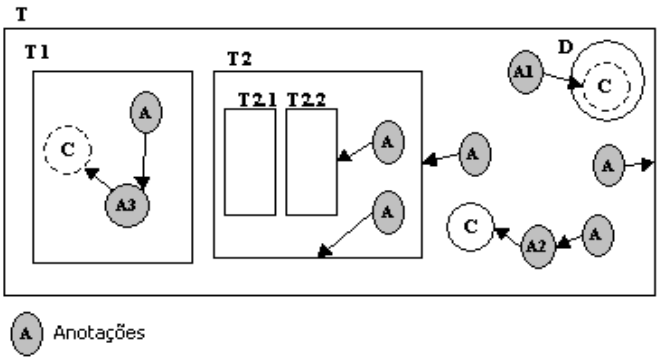


Figura 3.4: Anotações em tarefas.

⇒ Anotações contidas em *T* serão visíveis aos atores pertencentes às subtarefas *T1* e *T2*, e aos atores pertencentes a *T*.

⇒ Anotações contidas em *T* serão públicas e visíveis a *T* e a *T1* e *T2*, enquanto anotações contidas em *T1* são particulares a *T1*, e anotações contidas em *T2*, particulares à *T2* e às subtarefas contidas em *T2*.

Algumas considerações devem ser feitas a respeito da criação e visibilidade de anotações:

1. Como foi dito anteriormente, anotações devem ser criadas em tarefas apropriadas, quando permitido, levando-se em conta a visibilidade desejada.

2. Anotações poderão ser criadas, em uma tarefa *T*, por atores de T, sobre *T*, sobre subtarefas contidas em *T*, ou sobre documentos e anotações contidos em *T* ou em outra tarefa na qual *T* está recursivamente contida.

3. Anotações só poderão ser criadas em uma tarefa *T* (e fazer parte do *conjunto de anotações* de *T*) por atores pertencentes a outras tarefas que não *T* se o *autor* da anotação tiver permissão para inserir anotações no *conjunto de anotações* de *T*. A definição destas restrições é parte dos mecanismos de coordenação do trabalho.

4. Anotações não podem ser alteradas. Nesse sentido, uma nova anotação poderá ser criada sobre a anotação anterior que pode ser tornada obsoleta.

Anotações terão grande importância durante a realização de um trabalho, tanto na autoria assíncrona, como na autoria síncrona. Em ambos os casos, anotações possibilitarão que comentários e alterações em documentos ou tarefas possam ser destacados durante o processo, além de fornecer meios para a documentação de todo esse processo, no que diz respeito não só à alteração de documentos, mas também à comunicação entre seus atores e a tomadas de decisão.

3.2.2 NOTIFICAÇÃO

O controle de notificação será responsável pelo envio de notificações entre atores a respeito de determinados eventos ocorridos durante a realização do trabalho (realizado em

tarefas). Nesse sentido, o controle de notificações deverá se basear em um ***modo de cooperação*** definido para cada tarefa.

O *modo de cooperação*, definido para uma tarefa *T*, irá descrever se atores contidos em *T* trabalharão de maneira assíncrona, ou síncrona, ou se trabalharão em um modo fracamente acoplado ou fortemente acoplado.

O mecanismo de notificação será responsável pela definição de regras de notificação que irão proporcionar a consciência de um grupo de atores sobre a realização de certas atividades, de maneira adequada para o modo de cooperação no qual esse grupo irá cooperar.

No modo de cooperação fracamente acoplado será necessária apenas a propagação de notificações sobre alterações em dados compartilhados (documentos, tarefas, eventos em tarefas, anotações), enquanto no modo de cooperação fortemente acoplado[22] é necessária a propagação de notificações sobre alterações de interfaces que devem ser compartilhadas.

Baseado no modo de cooperação definido para cada tarefa poderiam ser definidas regras para a propagação de notificações entre atores pertencentes à tarefa. Todavia, a definição dessas regras não será apresentada neste trabalho, ficando como trabalho futuro.

3.3 COORDENAÇÃO

A estrutura de tarefas, elaborada para a realização do trabalho, a definição de papéis atribuídos a atores, e a definição de um Controle de Acesso irão caracterizar os mecanismos de coordenação que, auxiliados pelos mecanismos de comunicação, apresentados na seção anterior, possibilitarão a colaboração coordenada entre atores.

---

[22] A única restrição que deve ser feita sobre o modelo de tarefas, para que sejam respeitados os requisitos definidos por [HAAKE 1992], é que atores engajados em uma mesma tarefa, no modo de cooperação síncrono fortemente acoplado, estejam trabalhando com o mesmo conjunto de documentos. Esta restrição irá garantir a correta visibilidade de ações em interface compartilhada por tais atores.

Para que seja possível a definição de restrições de acesso a documentos ou tarefas durante o processo de autoria, faz-se necessária a especificação de um Controle de Acesso responsável pela coordenação das ações realizadas por atores de diferentes tarefas durante a realização do trabalho colaborativo.

Esta seção define como um Controle de Acesso pode ser utilizado como mecanismo de coordenação do trabalho realizado em tarefas.

Foram definidos para cada tarefa (onde o trabalho será realizado), um *Controle de Acesso para Tarefas* e outro *Controle de Acesso para Documentos*, responsáveis pela definição de regras de acesso a tarefas e documentos, respectivamente.

A definição de regras para a inserção ou remoção de documentos em uma tarefa *T*, inserção de anotações, alterações em estados de tarefa, e outras operações sobre atributos de tarefas, serão responsabilidade do *Controle de Acesso para Tarefas*, enquanto regras de acesso a documentos durante a autoria de documentos serão responsabilidades do *Controle de Acesso para Documentos*.

Deve-se notar que o termo *Controle de Acesso*, empregado genericamente até aqui, foi utilizado para representar um mecanismo que irá possibilitar o acesso controlado a um objeto. Nesse sentido, podem ser observadas duas funções básicas que devem ser oferecidas por esse mecanismo:

⇒ A definição de regras para o acesso concorrente a objetos; e,

⇒ a definição de *direitos de acesso* a indivíduos sobre objetos, onde permissões serão atribuídas a esses indivíduos de modo a representar, por exemplo, um direito de escrita em um objeto *A* por um indivíduo *X*, ou direito de leitura de um objeto *B* por outro indivíduo *Y*.

A partir de agora, o termo *controle de concorrência* será empregado para representar a primeira função, enquanto o termo *controle de acesso* será utilizado para representar a segunda função.

O *Controle de Acesso para Tarefas* e o *Controle de Acesso para Documentos* serão especializações de um *Controle de Acesso* genérico para Objetos, formado por um *controle de concorrência* e por um *controle de acesso* específico. A definição desse Controle de Acesso genérico fez parte deste trabalho e poderá ser vista no Capítulo 4.

Resumidamente, o *controle de acesso* especificado no *Controle de Acesso* genérico irá possibilitar o controle e definições de direitos de acesso a sujeitos[23] que desejam acessar um determinado objeto, assim como a definição de permissões atribuídas a esses sujeitos, que farão parte do direito de acesso definido; o *controle de concorrência* irá possibilitar a definição de travas em objetos concedidas a determinados sujeitos, restringindo um possível acesso concorrente a um objeto a apenas um sujeito, através de *travas* em objetos.

Objetos serão especializados em tarefas ou documentos, e sujeitos em atores. Quando objetos são especializados em tarefas, o Controle de Acesso genérico é especializado em um *Controle de Acesso sobre Tarefas*, e quando são especializados em documentos, o Controle de Acesso genérico é especializado no *Controle de Acesso sobre Documentos*.

A Figura 3.5 ilustra o modelo de *Controle de Acesso* para cada tarefa.

[23] Sujeitos, nesse caso, representam qualquer entidade que deseja realizar um acesso controlado a um objeto.

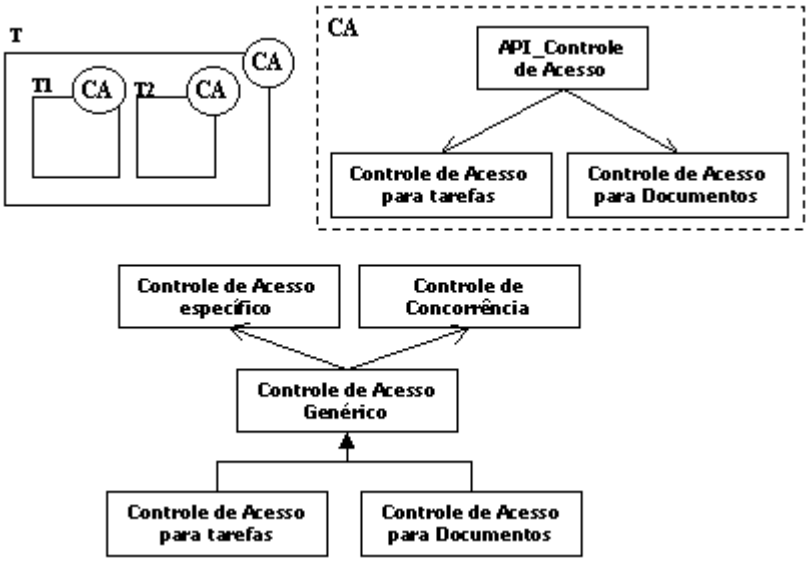


Figura 3.5: Controle de Acesso por tarefa.

O *controle de concorrência* definido para uma tarefa *T*, quando faz parte do *Controle de Acesso sobre Tarefas*, irá conter informações sobre travas relativas a *T*, enquanto o *controle de concorrência*, quando faz parte do *Controle de Acesso sobre Documentos* de uma tarefa *T*, irá conter travas relativas a documentos contidos em *T*. Através do *controle de concorrência* sobre documentos, será possível definir, por exemplo, um conjunto de documentos contidos em uma determinada tarefa que estão sendo alterados (no estado *temporário sendo alterado*).

O *controle de acesso específico* definido em uma tarefa *T*, quando faz parte do *Controle de Acesso sobre Tarefas*, irá permitir a definição de direitos de acesso para atores contidos em *T* sobre *T*, e, quando faz parte do *Controle de Acesso sobre Documentos*, irá permitir a definição de direitos de acesso para atores contidos em *T* sobre documentos contidos em *T* ou contidos em uma tarefa onde *T* está recursivamente contida.

A definição de um *Controle de Acesso* para cada tarefa (através do *Controle de Acesso sobre Documentos*) irá possibilitar a definição de diferentes *direitos de acesso* para um mesmo ator sobre um mesmo documento dependendo da tarefa onde o trabalho sobre o documento será realizado. Um documento, por exemplo, contido por recursão em uma tarefa *T1* deverá ser alterado conforme os direitos de acesso definidos em *T1*. No entanto, poderia ser que não fosse definido um direito de acesso para esse ator em *T1*, e,

nesse caso, o direito de acesso seria aquele definido na primeira tarefa de um aninhamento de tarefas onde *T1* estivesse recursivamente contida.

Vale notar que o Controle de Acesso apresentado nesta seção, e que será definido no Capítulo 4, especifica o uso de travas em objetos, como controle de concorrência entre atores. Todavia, o modelo de cooperação apenas irá determinar a necessidade de um controle de concorrência, assim como a necessidade de um controle de acesso, que deve possibilitar de alguma forma a definição de regras de acesso necessárias para a coordenação do trabalho.

3.4 MODELO DE COOPERAÇÃO

Depois de definidos os meios de colaboração, comunicação e coordenação, foi possível definir os componentes necessários para que o trabalho cooperativo pudesse ser realizado, baseado no modelo de tarefas apresentado no Capítulo 2. Esses componentes formarão o modelo de cooperação, apresentado no Seção 3.4.1.

Na seção 3.4.2, é apresentado um *modelo de objetos* para definir objetos granulares e possibilitar um maior grau de cooperação entre atores engajados em um processo cooperativo.

3.4.1 ESTRUTURA DO MODELO DE COOPERAÇÃO

Nesta seção, será apresentado o *modelo de cooperação*, que dará suporte a autoria cooperativa de documentos, através de um diagrama de classes em UML.

O modelo irá relacionar os componentes definidos ao longo do trabalho como atributos de tarefas, que irão proporcionar a definição dos mecanismos apresentados como suporte à colaboração, comunicação e coordenação, necessários ao trabalho cooperativo. Esse modelo terá como componente básico a entidade *tarefa*, onde o trabalho deverá ser realizado, e para qual foram determinados certos atributos.

A Figura 3.6 descreve a estrutura básica do modelo de cooperação.

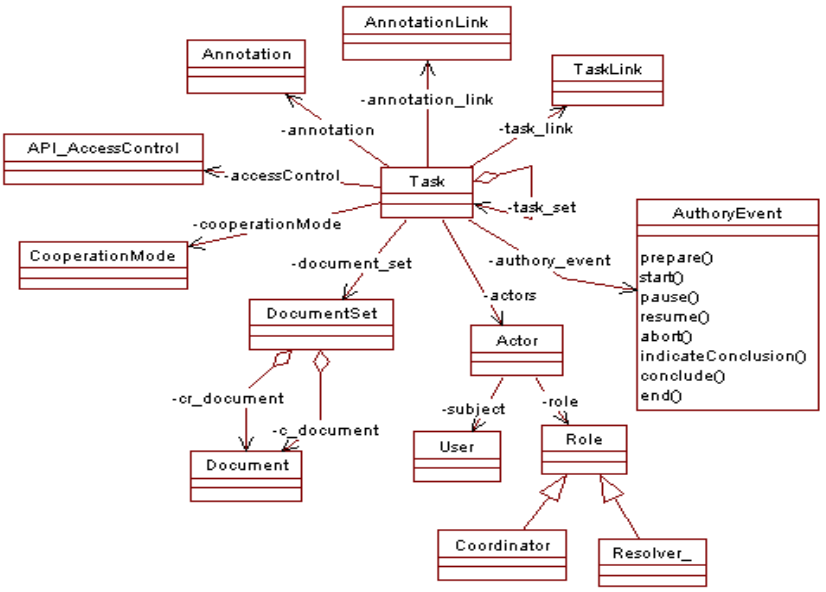


Figura 3.6: Componentes de tarefas.

Tarefas deverão conter como atributos básicos: um conjunto de subtarefas (*subtasks*); um conjunto de relacionamentos entre tarefas (*task_link*); um conjunto de atores (*actors*), com seus respectivos papéis; o modo de cooperação, segundo o qual atores de tarefas irão cooperar (*cooperationMode*); o *evento de autoria* (*authory_event*); um conjunto de documentos (*document_set*); um conjunto de anotações (*annotation*); um conjunto de relacionamentos entre anotações, que descrevem o relacionamento entre anotações e quaisquer objetos que podem ser anotados – tarefas documentos ou anotações – (*annotation_Link*); e, um controle de acesso (*accessControl*).

Cada uma das classes na figura representa componentes do modelo, que devem ser especificados de modo a descrever as características apresentadas para cada um ao longo desta dissertação. Em relação a isso alguns comentários devem ser feitos.

O *conjunto de documentos* (*document_list*) foi modelado de modo a conter os documentos utilizados durante o trabalho em *T*. Esse conjunto irá conter os documentos contidos em uma tarefa (*c_documents*), assim como os documentos *contidos por recursão* (*cr_documents*), modelados como subconjuntos de um *conjunto de documentos* (*document_set*) de uma tarefa.

Como dito anteriormente, um ator (*Actor*) descreve um papel (*role*) que deve ser atribuído a um usuário (*User*) para determinar restrições e permissões relativas a um

determinado indivíduo, participante da autoria em uma determinada tarefa. No caso, dois papéis foram definidos (*coordinator* e *resolver*), como o foram na Seção 2.2. Esses papéis, definidos para atores de uma tarefa *T* poderiam agir junto ao Controle de Acesso (*accessControl*), definido em T, para definir diretamente permissões sobre tarefas ou documentos, dependendo da semântica atribuída a cada papel. Nesse sentido, papéis agiriam como facilitador durante a definição de responsabilidades a cada ator de uma determinada tarefa.

Foi definida a classe *AuthoryEvent*, representando o evento de autoria, onde serão definidos os possíveis estados do *evento de autoria* e as ações que poderão ser realizadas em cada estado do evento, de acordo com a máquina de estados definida em 2.2.2, e as ações definidas em 2.2.3 [24].

O diagrama para o controle de acesso, ilustrado na figura seguinte, segue a lógica apresentada na Seção 3.3.

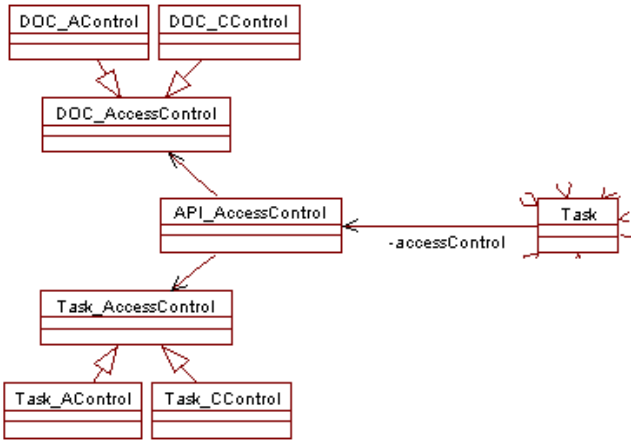


Figura 3.7: Controle de Acesso para tarefas.

[24] As definições do evento de autoria e da máquina de estados para o evento de autoria poderiam ser modeladas seguindo-se o pattern State [GAMMA], todavia achou-se que a definição de todas as classes para a representação dos possíveis estados do evento não traria grandes vantagens para o modelo.

3.4.2 MODELO DE OBJETOS

Como foi dito anteriormente, a cooperação entre indivíduos envolve o compartilhamento de informações de maneira coordenada, evitando que indivíduos se envolvam em tarefas conflitantes. O controle de concorrência baseado em travas, que será utilizado no modelo de cooperação apresentado, irá agir nesse sentido, evitando que indivíduos alterem um mesmo objeto concorrentemente. A forma como objetos componentes do modelo são definidos irá influenciar em muito no grau de compartilhamento de informações.

O *modelo de objetos genéricos*, que será apresentado nesta seção, irá permitir a definição de objetos granulares como meio de possibilitar uma maior cooperação entre atores. O *modelo genérico de objetos* irá definir objetos como uma composição de outros objetos.

Objetos poderão ser objetos simples, quando não contém outros objetos, ou objetos compostos, quando contém objetos simples ou outros objetos compostos. Esses objetos poderão representar quaisquer entidades formadas por atributos, que poderiam ser por sua vez atributos simples ou atributos compostos (seguindo o modelo tradicional de orientação a objetos). É nesse sentido que o modelo de objetos será útil na definição dos componentes do modelo de cooperação. Todos os componentes (objetos) serão vistos como *Entidades*, e seus atributos (também objetos) serão chamados no modelo de *Propriedades*.

A Figura 3.8 ilustra o *modelo de objetos genéricos*, especializado para *Entidades* e *Propriedades*, segundo o diagrama de classes em UML:

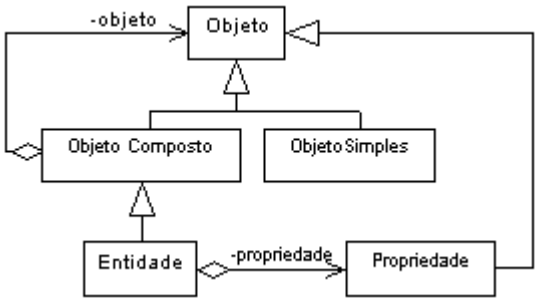


Figura 3.8: Modelo genérico de objetos.

As classes *Objeto, Objeto Simples* e *Objeto Composto* são relacionadas segundo o pattern *Composite* definido em [GaHJV 1994].

A classe *Propriedade* foi definida como uma especialização da classe *Objeto*, podendo ser especializada em *Objeto Simples* ou *Composto*. No segundo caso, atributos poderão conter outros atributos. A classe *Entidade* foi criada como uma especialização de *Objeto Composto*, formado pela composição de outros *Objetos*, mais especificamente, *Propriedades*.

Tarefas, assim como documentos e outros componentes do modelo de cooperação, serão definidos como especializações de *Entidade*, cada um com seus atributos básicos definidos (*Propriedades*). Atributos de tarefas ou documentos, definidos como *Propriedades*, serão manipulados como objetos do modelo, e por isso, poderão ser tratados de forma independente pelos mecanismos de coordenação durante a autoria cooperativa, mais especificamente pelo *controle de concorrência*.

Seguindo esse modelo, poderão ser compartilhados atributos de objetos (*Propriedades*), assim como poderão ser definidas regras de acesso independentes para cada atributo de uma tarefa ou documentos. Torna-se possível, dessa maneira, a autoria cooperativa sobre atributos de tarefas ou sobre atributos de documentos.

É importante observar que determinadas restrições definidas em papéis, como a restrição para se criar ou remover subtarefas contidas uma tarefa específica *T*, ou a possibilidade de remover ou adicionar atores em *T*, pode ser vista, através desse modelo de objetos, como permissões sobre atributos de *T*. No caso sobre o conjunto de subtarefas de *T* e sobre o conjunto de atores de *T*, respectivamente.

Ao longo deste capítulo foram apresentados os mecanismos que darão suporte à colaboração, comunicação e coordenação do trabalho durante o processo cooperativo. Para que esses mecanismos efetivamente dêem suporte ao trabalho cooperativo, devem ser suportados por outros mecanismos como dito anteriormente. Nesse sentido, um controle de notificação será essencial para o suporte à comunicação entre os participantes de um grupo de trabalho, seja durante a colaboração entre autores, seja durante a coordenação do

trabalho, e deve ser disponibilizado para que atores possam ser notificados de determinados eventos. Ainda, para possibilitar o suporte à autoria semi-síncrona deve ser implementado um controle sobre versões de documentos ou mesmo sobre tarefas. Em todos os casos um controle de acesso será fundamental para possibilitar a coordenação do trabalho que será realizado em tarefas.

Os dois primeiros controles não serão apresentados neste trabalho, mas devem estar presentes durante um processo cooperativo. As referências [SoSRM 1999] e [Muniz 2000] trazem uma discussão e definição desses controles, de versão e notificação respectivamente, para o sistema HyperProp.

O controle de acesso faz parte deste trabalho e será apresentado no próximo capítulo.

4. CONTROLE DE ACESSO A OBJETOS

O Controle de Acesso irá possibilitar a coordenação do trabalho durante a autoria de documentos ou tarefas através da definição de regras de acesso a documentos ou tarefas, respectivamente.

Neste capítulo, serão apresentados os controles de acesso e concorrência genéricos para objetos de maneira geral. Esses controles, especializados para tarefas e documentos, farão parte do Controle de Acesso que deverá ser oferecido como mecanismo de coordenação necessário durante o trabalho cooperativo.

É importante observar que não serão tratados neste capítulo mecanismos de autenticação. O Controle de Acesso irá tratar de um *Sujeito* já certificado por um controle de autenticação, se isso for necessário ao sistema.

4.1 CONTROLE DE ACESSO

O controle de acesso diz respeito às permissões que um determinado indivíduo ou grupo de indivíduos possui sobre um determinado objeto.

Nesta seção, será apresentada uma arquitetura genérica para a definição de um controle de acesso que deverá definir basicamente relacionamentos de permissão sobre objetos por determinados indivíduos. Mais especificamente, será definido um *Framework* [FaySJ 1999], orientado a objetos, para Controle de Acesso, que deve oferecer as características fundamentais para que se permita acesso controlado a objetos.

De maneira geral, o Controle de Acesso definirá um relacionamento de permissão entre dois elementos básicos:

- o sujeito que deseja realizar o acesso (*Subject* - *Sujeito*) a um determinado objeto;
- o objeto, ao qual será realizado o acesso (Object - *Objeto*).

A permissão entre *sujeitos* e *objetos* também será representada por um objeto, o objeto *permissão* (*Permission – Permissão* de acesso). *Objetos* poderão ser qualquer tipo de entidade sobre a qual o acesso será controlado, e *sujeitos* qualquer tipo de entidade que deseja acessar um *objeto.*

A arquitetura genérica do Controle de Acesso será dividida em dois módulos de acordo com as seguintes responsabilidades:

⇒ O módulo Controle de Acesso definirá relacionamentos entre *sujeitos, objetos e permissões*. Será responsável por consultas de acesso, por criar e remover relacionamentos e alterar permissões;

⇒ e, o módulo Controle de Permissões tratará da definição das permissões que poderão fazer parte dos relacionamentos definido pelo módulo Controle de Acesso. Esse módulo possibilitará a definição de um conjunto de permissões próprio para cada objeto de um determinado sistema.

Esses dois módulos devem se relacionar para verificar a consistência das *permissões* que podem ou não ser atribuídas a um *sujeito* sobre um *objeto.*

Uma interface irá oferecer as funções do Controle de Acesso e do Controle de Permissão. Os relacionamentos entre esta interface e os módulos podem ser vistos a seguir, na Figura 4.1.

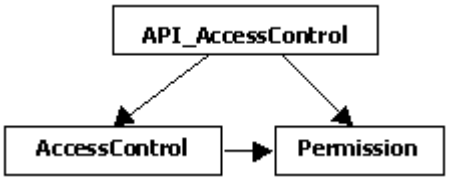


Figura 4.1: Arquitetura genérica do Framework de Controle de Acesso.

A API fornece as funcionalidades do Framework e funciona como uma fachada (*pattern Facade* [GaHJV 1994]) escondendo os módulos internos do Framework.

Os módulos *AccessControl* e *Permission* representam o módulo Controle de Acesso e Permissão, respectivamente, e que serão detalhadamente definidos a seguir.

4.1.1 MÓDULO CONTROLE DE ACESSO

O módulo Controle de Acesso rege o relacionamento entre os elementos básicos (*sujeito, objeto e permissão*) do sistema. Esse módulo faz o papel de um gerente de objetos do tipo *Direito de Acesso* e possui conhecimento de todos os possíveis acessos de *sujeitos* sobre *objetos* do sistema.

O Framework especifica uma agregação simples de *sujeito*, *objeto* e *permissão*, para compor um objeto *Direito de Acesso* (*AccessRight*). *Objetos, Sujeitos e Permissões* são *pontos de flexibilização* do modelo que deverão ser especializados.

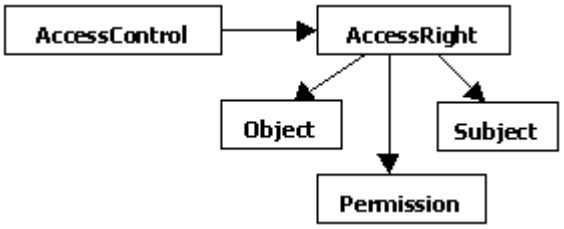


Figura 4.2: Modelo para o módulo Controle de Acesso.

De maneira geral, o *Controle de Acesso* (*AccessControl*) possui objetos *Direitos de Acesso* (*AccessRight*), representando um acesso de um *sujeito* sobre um *objeto*, segundo uma *permissão*, e sabe como criá-los, alterá-los e removê-los.

O Controle de Acesso deve fornecer algumas funcionalidades básicas que possibilitem:

- criar novos acessos;
- remover acessos existentes;
- consultar um determinado acesso de um sujeito sobre um objeto;
- alterar acessos existentes, se for possível;
- e, retornar a permissão de um sujeito sobre um objeto, se existir.

Novas funcionalidades poderão ser adicionadas ao *Framework*, como a possibilidade de verificar todos os sujeitos que tem permissão sobre um determinado objeto, ou sobre que objetos um determinado sujeito tem permissão.

4.1.2 MÓDULO CONTROLE DE PERMISSÃO

O Módulo Controle de Permissão define as permissões que objetos em um sistema que utiliza o Controle de Acesso pode suportar. Estas *permissões* são objetos que possuem um *nome*, uma *descrição* e um *conjunto de ações* que um *sujeito* com permissão sobre um *objeto* pode realizar. *Ações*, por sua vez, também são objetos que possuem um *nome* e uma *descrição*.

Um sistema pode definir, através do Controle de Permissão, o *conjunto de permissões* possíveis a um determinado objeto, podendo deixar em aberto a inserção de novas permissões ou não em tempo de execução do sistema[25]. O mesmo conjunto de permissões pode ser atribuído a mais de um tipo de objeto, assim como se podem definir conjuntos de permissões distintos para cada tipo de objeto do sistema[26].

Para cada *conjunto de permissões P* deve ser definido o *conjunto de ações* possíveis que podem ser atribuídas a uma *permissão* do conjunto *P*.

Conjunto de permissões e ações constituem *pontos de flexibilização* do Framework que devem ser especializados para compor o conjunto de permissões possíveis a determinados indivíduos sobre um determinado tipo de objeto.

A Figura 4.3 apresenta um modelo para o Controle de Permissões.

[25] Permissões não poderão ser removidas durante a execução do sistema, de modo a evitar inconsistências de direitos de acesso.

[26] Objetos podem ser arquivos de programa, diretórios, ou qualquer tipo de objetos do sistema que necessitem de tratamento diferenciado em algum momento.

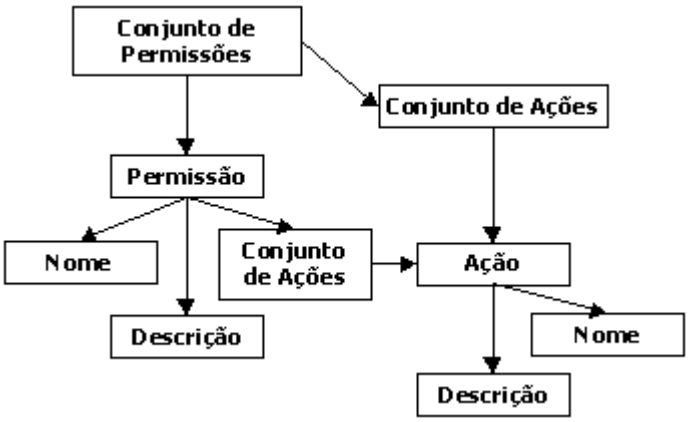


Figura 4.3: Modelo para o módulo de Permissões.

O relacionamento entre o *conjunto de permissões* e o *conjunto de ações* possíveis possibilita a verificação de consistência quando uma permissão é definida.

Assim como o Controle de Acesso, o Controle de Permissão deve fornecer algumas funcionalidades básicas que permitam:

- Adicionar uma nova permissão ao conjunto de permissões possíveis a um objeto se esse conjunto de permissões estiver configurado para isso[27];
- Retornar as permissões possíveis para um determinado objeto;
- Verificar se uma determinada ação pode ser realizada por uma permissão determinada;
- Adicionar uma nova ação a um conjunto de ações possíveis a um objeto, se esse conjunto estiver configurado para isso;
- Retornar as ações possíveis de serem realizadas sobre um objeto;
- Retornar as ações que podem ser realizadas por uma permissão;

Se for necessário, essas funcionalidades podem ser estendidas de modo a atender as necessidades de um sistema particular.

[27] Os conjuntos de permissão e ação podem ser configurados como "read-only", quando novas permissões e ações, respectivamente, não poderão mais ser inseridas.

O *módulo controle de permissão* possibilitará a definição de permissões e ações possíveis a objetos, independente do *módulo controle de acesso*, o que tornará o Controle de Acesso ainda mais genérico.

Deve-se notar que da maneira como foi definido o *controle de acesso*, esse poderá ser acresentado a qualquer sistema que deseja acrescentar características de segurança em relação a seus dados (objetos, segundo um modelo orientado a objetos), sem, no entanto, ter que alterar o código dos objetos já existentes.

A idéia é que antes que chamadas a métodos de objetos (que descrevem as ações que podem, a princípio, ser realizadas por objetos) sejam feitas, se faça uma chamada ao controle de acesso, quando é feita uma verificação sobre a permissão do usuário sobre a realização da operação sobre o objeto. Tendo permissão, o usuário poderá relizar a ação desejada (quando o método adequado poderá ser executado).

Pode-se dizer que, basicamente, o controle de acesso irá definir permissões sobre métodos de objetos, determinando quais métodos (ações) poderão ser executados por objetos. Como foi especificado na arquitetura do *módulo de controle de permissão*, o *conjunto de ações* de um objeto foi definido como componente de uma pemissão, e irá definir as ações (métodos) que poderão ser realizadas por um usuário. O objeto *permissão* foi especificado de modo a fornecer uma interface adequada para usuários do controle de acesso (que não precisam ter a noção de quais métodos poderão ser executados por ele, apenas uma noção de mais alto nível e mais explicativa, como uma permissão para a alteração do conteúdo de um documento, por exemplo), além de possibilitar a definição de permissões sobre métodos de um objeto de forma integrada (permissão sobre um conjunto de métodos). A definição de *permissões*, que poderão ser atribuídas a um usuário sobre um determinado objeto, deverá ser responsabilidade de um administrador do sistema, que possui conhecimento dos objetos e métodos que devem ser controlados.

Dependendo do propósito de cada sistema, uma permissão poderá ser definida de maneira a conter ações que não poderão ser realizadas por um usuário sobre um objeto, ao invés das ações que poderão ser realizadas. Em ambas as formas um teste deverá ser feito,

antes da chamada a um método do objeto, para verificar a possibilidade de execução de um método por um usuário.

Deve-se observar que o uso do controle de acesso como um mecanismo independente dos objetos que deverão ser controlados em um sistema, apesar de oferecer a vantagem de ser facilmente adaptável a sistemas que desejem rapidamente acrescentar características de controle de acesso, possui desvantagens em relação àqueles que já oferecem o controle de acesso como uma característica interna dos objetos de um sistema (como, por exemplo, embutido no próprio método do objeto).

No primeiro caso, para todas as chamada a métodos de um objeto, um desvio deve ser feito ao controle de acesso para se verificar a permissão de um usuário em relação à chamada do método, enquanto no segundo caso, uma chamada ao controle de acesso só seria feita quando fosse necessário (quando houvesse uma restrição explícita sobre a execução de um determinado método). Outra desvantagem diz respeito ao controle sobre chamadas de métodos por outros métodos, quando pode acontecer de ser definida uma permissão para a execução de um determinado método que chama outro método sobre o qual o usuário não teria permissão para executar (o que não teria problemas se um teste fosse feito diretamente pelo método a ser executado). Empregando o controle de acesso genérico, deve-se preocupar com a definição de permissões, de maneira que o conjunto de ações de uma permissão seja definido de forma coerente.

4.2 CONTROLE DE CONCORRÊNCIA

O Controle de Concorrência, que será apresentado nesta seção, irá impossibilitar o acesso a um objeto genérico por mais de um indivíduo, como forma de evitar uma possível inconsistência em um trabalho concorrente.

Essa proposta, embora possa não parecer a mais adequada para trabalhos cooperativos, se torna interessante para a autoria de documentos hipermídia, à medida que documentos poderão ser definidos como uma composição de documentos mais granulares, interligados por elos. O Controle de Concorrência atuaria nesses documentos,

possibilitando a colaboração entre indivíduos, sem, no entanto, inserir a complexidade de outras possíveis soluções para o controle de acessos concorrentes.

Uma outra maneira de aumentar a granularidade da informação compartilhada por indivíduos, é permitir que diferentes atributos de um mesmo objeto possam ser compartilhados por diferentes indivíduos, como foi apresentado na Seção 3.4.2. Como foi mostrado, isso irá depender do modelo de objetos utilizado para a definição do *objeto* que deverá ter seu acesso controlado.

O Controle de Concorrência proposto será baseado em um esquema de travas, e deve ser consultado na tentativa de alteração de um objeto. Da mesma maneira que o Controle de Acesso definido na Seção 4.1, o Controle de Concorrência será modelado de modo a reger o acesso a objetos genéricos por indivíduos.

Objetos poderão ser especializados em tarefas ou documentos para que sejam definidos os Controles de Acesso a tarefas e documentos, como apresentado na Seção 3.3.

CONTROLE POR BLOQUEIO

As travas representam bloqueios realizados em objetos por um sujeito. Como no Controle de Acesso, *objetos* representarão qualquer entidade que necessite ter seu acesso controlado, e *sujeitos* representarão um indivíduo, devidamente autenticado, que deseja realizar um acesso sobre o *objeto.*

Travas serão atribuídas a um *sujeito* sobre um *objeto* mediante um pedido explícito ao Controle de Travas, e poderão ser suspensas quando o *sujeito* ou outro indivíduo autorizado fizer outro pedido explícito para sua suspensão.

Algumas questões sobre *travas* foram analisadas, ainda, para que o modelo para o Controle de Concorrência por travas fosse criado:

⇒ Quando dois usuários tentam obter uma *trava* de um determinado objeto, apenas um o obtém. Deve-se armazenar o outro pedido em uma fila de *travas*?

⇒ Filas de *travas* devem ser levadas em conta, ou não?

⇒ Deve-se prover mecanismos para notificar quando um nó de interesse se torna livre (*destravado*)?

⇒ O sistema deve prover mecanismos para negociação de *travas*?

⇒ Deve ser possível que um usuário "roube" a *trava* de outro usuário, sobre um determinado *objeto*?

Travas serão armazenadas em *filas de pedidos* por *objeto* de acordo com alguma estratégia definida. A primeira *trava* da *fila de pedidos* de um *objeto* irá bloquear o objeto para o *sujeito* que fez o pedido de bloqueio, se esse não estiver bloqueado. Quando isto acontece, a *trava* deixa de fazer parte da *lista de pedidos*, e passa a estar atribuída a um *sujeito.*

O *objeto* poderá ser desbloqueado mediante a um pedido explícito de desbloqueio por um *sujeito* autorizado. Da mesma maneira, uma *trava* poderá ser removida da *fila de pedidos* por um *sujeito* autorizado.

Uma trava pode ser armazenada no *conjunto de pedidos* por ordem de chegada ou por outras prioridades, dependendo de uma estratégia definida para o consentimento de travas. Cada estratégia deve ser implementada em classes de estratégia específicas, especializações de uma estratégia genérica que oferece a interface para a inserção de pedidos na *fila de pedidos*. Estratégias serão responsáveis somente pela inserção em uma ordem específica de *travas* no *conjunto de pedidos*. O modelo de estratégias segue o pattern Strategy exposto em [GaHJV 1999].

Como meio de possibilitar a consciência por atores da alteração de documentos, travas sobre *objetos* serão visíveis para o sistema de autoria. Nesse sentido, não será criado um mecanismo explícito de notificação de atores sobre o desbloqueio de *travas*, mas será evidente quando isto ocorrer, já que o objeto não mais fará parte da lista de travas. É necessário notificar, no entanto, a atribuição de travas a sujeitos como pedidos no *conjunto de pedidos*, para que esse possa começar a realizar o trabalho desejado.

O modelo de travas pode ser visto na Figura 4.4. Detalhes de especificação e implementação do modelo podem ser vistos no Apêndice A.

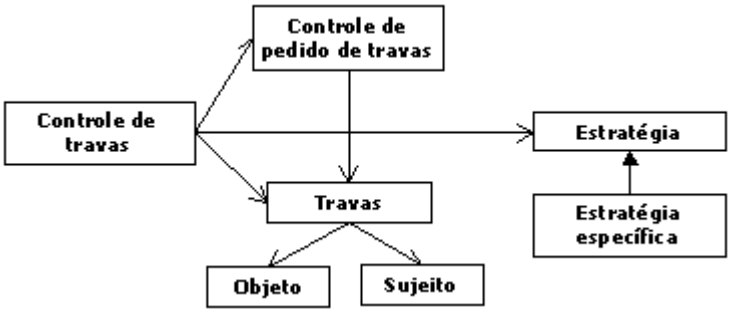


Figura 4.4: Modelo para controle de travas.

O Controle de Travas irá oferecer a interface para o Controle de Concorrência e deve fornecer as seguintes funções:

- Criar travas em *objetos* por *sujeitos*;
- Remover travas;
- Testar se uma trava existe;
- Retornar travas em objetos;
- Retornar travas na *fila de pedidos*.

O Controle de Concorrência é um controle genérico, à medida que poderão ser realizadas *travas* em qualquer *objeto*, com tanto que a interface de *Objeto* seja respeitada. Do mesmo jeito, qualquer entidade que respeitar a interface *Sujeito* pode obter uma *trava* sobre um *objeto*. Além disso, o controle permite que estratégias de atribuição de travas sejam estabelecidas, o que pode depender do *objeto* que deve ser bloqueado. Por isso, as classes *estratégia*, *objetos* e *sujeitos* podem ser considerados *pontos de flexibilização* de um *framework* para controle de concorrência baseado em travas.

Assim como travas em objetos, pedidos de trava são armazenados em uma fila por objeto, de maneira que possam ser vistos pelos usuários conectados ao sistema. Isto permite que usuários possam estar cientes do estado de travas de cada objeto, possibilitando a detecção de possíveis *deadlocks*. Vale notar que não foi implementada uma política implícita de detecção e correção de *deadlocks*, e por isso fica a cargo de

atores especializados a sua detecção e correção. Nesse caso, a correção de *deadlocks* deverá ser realizada explicitamente, através de um pedido explícito de desbloqueio, ou de remoção de travas no *conjunto de pedidos* de travas. No entanto, poderia ser criada uma estratégia para a inserção de travas no *conjunto de* pedidos que se preocupasse com a existência de *deadlocks*, o que, todavia, aumentaria em muito a complexidade da estratégia.

O Controle de Acesso definido de forma genérica, neste capítulo, deverá ser especificado para atender as necessidades de coordenação citadas na Seção 3.3.

Nesse sentido, tarefas e documentos deverão seguir a interface de *Objeto* determinada no Controle de Acesso, para compor o Controle de Acesso a Tarefas e o Controle de Acesso a Documentos. Ainda, atributos de tarefas ou documentos poderão ser definidos como *Objetos*, de maneira a possibilitar um maior grau de cooperação entre atores, como apresentado em 3.4.2. Atores deverão seguir a interface *subject*, assim como permissões terão que ser definidas para tarefas e documentos, ou mesmo para atributos de *Objetos*.

Uma instância do Framework definido neste capítulo pode ser vista no Apêndice A, quando é definido o Controle de Acesso para objetos NCM.

5. COOPERAÇÃO NO HYPERPROP

Neste capítulo, o modelo de cooperação, apresentado no Capítulo 3, será adaptado ao sistema HyperProp (na Seção 5.3), assim como será apresentado um ambiente para a manipulação de determinados componentes definidos no modelo de cooperação, essenciais para a autoria cooperativa (na Seção 5.4).

Inicialmente, o *modelo de tarefas*, apresentado em 2.2, será especializado em um *modelo de tarefas hipermídia*, estruturado segundo certos conceitos usuais em sistemas hipermídia (na Seção 5.1), e mais especificamente segundo conceitos definidos no NCM, para formar o *modelo de tarefas* do HyperProp. Sobre essa estrutura especializada, serão apresentados os mecanismos que irão possibilitar a colaboração, comunicação e coordenação no HyperProp, e formar o *modelo de cooperação hipermídia* definido para esse sistema.

Como esse modelo irá utilizar objetos NCM para a sua composição, algumas definições relevantes, nesse sentido, ainda serão apresentadas na Seção 5.2, definições essas, que serão discutidas durante a Seção 5.3, para compor o modelo de cooperação.

5.1 MODELO DE TAREFAS HIPERMÍDIA

O *modelo de tarefas hipermídia* pode ser visto como uma especialização do *modelo de tarefas*, definido na Seção 2.2, se forem acrescentados, a esse, certos conceitos e características usuais em sistemas hipermídia.

A idéia é criar um modelo que irá possibilitar, mais facilmente, o acréscimo de determinadas características de cooperação, oferecidas pelo *modelo de tarefas*, a sistemas hipermídia, ao mesmo tempo em que certas funcionalidades oferecidas por esses sistemas hipermídia serão acrescentadas ao modelo de tarefas.

Basicamente, o *modelo de tarefas hipermídia* será definido utilizando-se os conceitos de nós e elos hipermídia, para representar tarefas e relacionamentos entre tarefas,

mantendo-se as definições apresentadas na Seção 2.2, e de acordo com as seguintes definições:

- *Nós*, tradicionalmente, contém um conjunto de unidades de informação, que podem ser interligados por elos. *Nós de composição* são, de maneira geral, nós hipermídia que podem conter outros nós.
- *Elos* representam o relacionamento entre nós, e, dependendo da sua definição em um modelo conceitual hipermídia específico, poderão determinar regras de relacionamentos entre nós mais ou menos elaboradas.

Mais especificamente, tarefas serão representadas por uma especialização de *nós de composição*, uma vez que, a princípio, tarefas poderão conter outras tarefas[28], além de documentos.

As definições de tarefas e relacionamentos irão depender de um modelo conceitual hipermídia específico, que irá determinar as especificações de nós e elos, e, conseqüentemente, uma especificação do *modelo de tarefas* mais ou menos imediata. De qualquer forma, o *modelo de tarefas* poderá ser visto como uma extensão do próprio modelo conceitual, já que tarefas e relacionamento serão especializações de nós e elos.

A definição do *modelo de tarefas* segundo o modelo conceitual de dados hipermídia NCM será apresentada, na Seção 5.3, e utilizada como suporte ao *modelo de cooperação*, que também será apresentado naquela seção.

5.2 O NCM – ALGUMAS DEFINIÇÕES RELEVANTES

O Modelo de Contextos Aninhados - NCM (*Nested Context Model*) é um modelo conceitual de dados hipermídia, orientado a objetos, cujo modelo de interface separa os componentes de dados e de exibição dos objetos, permitindo a manipulação de documentos independente da plataforma final de exibição.

---

[28] A criação de novas subtarefas em uma determinada tarefa *T* poderá ser restringida pela definição de direitos de acesso em *T*.

Original na definição de composições aninhadas [CTLNRS 1991], a definição de documentos hipermídia no NCM é baseada nos conceitos usuais de nós e elos, onde *nós* são fragmentos de informação e *elos* são usados para a definição de relacionamentos entre os nós que interligam. O modelo distingue duas classes de nós chamados de *nós terminais*, ou de conteúdo, e *nós de composição*, sendo esses últimos o ponto central do modelo.

Alguns dos objetos que compõem o modelo serão brevemente apresentados nesta seção. Detalhes sobre o modelo podem ser vistos em [Soare 2000].

5.2.1 MODELO DE OBJETOS NCM

Objetos NCM são definidos como *entidades*, que contém determinados atributos (ou *propriedades NCM*).

Para possibilitar a manipulação de objetos NCM de mais fina granularidade, principalmente durante à autoria de documentos, *propriedades NCM* foram modeladas como objetos, assim como *entidades* (seguindo o modelo tradicional de orientação a objetos).

A idéia é muito semelhante à definição de objetos segundo o *modelo de objetos genéricos*, apresentado na Seção 3.4.2, todavia, com uma restrição: a definição de *propriedades NCM* não permite a sua composição por outras *propriedades NCM*. Essa definição facilita em muito a implementação do modelo, mas poderá trazer algumas restrições à cooperação, já que atributos de uma propriedade não poderão ser compartilhados ou tratados de forma independente. Atores, nesse caso, não poderão manipular diferentes atributos de propriedades de forma independente.

O *modelo conceitual de objetos NCM*, dessa maneira, pode ser visto como uma especialização do *modelo genérico de objetos*, onde *objetos NCM* são especializações de *Entidade*, compostas por *Propriedades*, representando seus atributos, que, no entanto, não poderão ser compostas por outras propriedades.

5.2.2 HIERARQUIA DE CLASSES NCM

A Figura 5.1 ilustra a hierarquia das classes que serão utilizadas de alguma forma para compor o modelo de cooperação, que será definido na Seção 5.3. Essas classes constituem um subconjunto daquelas que compõem o modelo conceitual NCM, e serão resumidamente apresentadas em seguida.

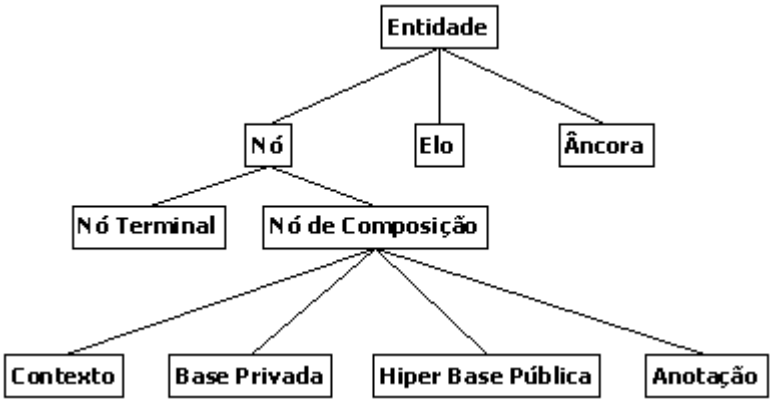


Figura 5.1: Estrutura hierárquica de objetos NCM.

Toda *entidade* do modelo NCM possui como atributos básicos: um identificador único (UID), um nome, uma descrição textual da entidade, a hora e data de criação e um usuário, responsável pela criação da entidade (denominado dono da entidade). Para cada atributo, propriedade NCM, a entidade NCM oferece suporte para manter as informações do último usuário que alterou seu valor e da data e hora em que essa modificação foi efetuada.

5.2.2.1 NÓS E ÂNCORAS

Um ***nó*** é uma entidade NCM que tem como atributos básicos adicionais um *conteúdo* e um *conjunto de descritores* alternativos.

O *conteúdo* de um nó é composto por uma coleção de unidades de informação, que, por exemplo, poderão ser um conjunto de caracteres ou um conjunto de quadros de vídeo, dependendo do tipo de conteúdo do nó.

O *conjunto de objetos descritores* alternativos contém um conjunto de descritores dos quais um deve ser (ou foi) selecionado (a escolha depende de parâmetros definidos na plataforma de exibição onde o documento estiver sendo exibido), ou o valor nulo. O

descritor selecionado de um nó contém informação determinando como o nó é apresentado, no tempo e no espaço, ou como deve ser comportar em reação a uma determinada ação realizada sobre o nó.

Um ***nó terminal*** *(*ou *nó de conteúdo)* é um nó que possui como atributos básicos adicionais: a *especificação do tipo de conteúdo*, uma lista ordenada de *âncoras* e o *estado*[29] do nó. Nós terminais são nós cujo conteúdo é dependente da aplicação. O modelo permite que a classe de nós terminais seja especializada em outras classes (texto, áudio, imagem, etc).

O atributo *especificação do tipo de conteúdo* contém uma descrição precisa dos dados armazenados no atributo conteúdo herdado da classe nó. É a especificação do tipo de conteúdo que vai permitir que se saiba qual ferramenta deve ser invocada para tratar o conteúdo do nó.

Uma ***âncora*** define um segmento, denominado *região*, dentro do conteúdo de um nó. A região pode ser um conjunto de unidades de informação ou o conteúdo como um todo, quando será chamada de âncora λ. A referência a uma âncora será geralmente feita pela identificação do nó (UID do nó) e o nome da âncora ou sua posição na lista de âncoras.

Um ***nó de composição*** *C* é um nó cujo *conteúdo* é uma coleção C_L de nós (de conteúdo e de composição, recursivamente) que se constituem em suas unidades de informação. Diz-se que um nó *N* em C_L é um *componente de C* e que *N está contido em C*. Diz-se também que um nó *A* está *recursivamente contido em C* se e somente se *A* está contido em *C* ou *A* está contido em um nó recursivamente contido em *C*. Um nó pode estar contido mais de uma vez em C_L, mas uma restrição importante é feita: um nó não pode estar recursivamente contido em si mesmo.

[29] A definição de estado de um nó terminal poderá ser vista mais adiante, quando for apresentada a definição de nós de contexto.

Nós de composição diferentes podem conter um mesmo nó e nós de composição podem ser aninhados em qualquer profundidade, desde que a restrição de um nó não conter recursivamente a si mesmo seja obedecida. Para identificar através de que seqüência de nós de composição aninhados uma dada instância de um nó *N* está sendo observada, é introduzida a noção de perspectiva de um nó. A *perspectiva* de um nó *N* é uma seqüência P = (N_m,....,N_1), com $m > 1$, tal que $N_1 = N$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para i ? [1,m) e N_m não está contido em qualquer nó. Poderá haver várias perspectivas diferentes para um mesmo nó *N*, se esse nó estiver contido em mais de uma composição.

Um ***nó de contexto*** *de usuário* (ou, simplesmente, *nós de contexto*) *U* é um nó de composição cujo conteúdo é um conjunto *S* de *nós terminais* e *nós contextos de usuário*. O termo *documento NCM* é utilizado para referenciar indistintamente nós terminais e de contexto. Dessa forma, um nó de contexto é um documento NCM que contém outros documentos NCM. Os nós de contexto possuem como atributos básicos adicionais: uma *lista ordenada de âncoras*, um *conjunto de elos*, uma *coleção de apresentação* e um *estado*. Nós de contexto vão servir, entre outras coisas, para definir uma estrutura, hierárquica ou não, para documentos, para permitir a definição de diferentes visões de um mesmo documento e para melhorar a orientação do usuário na navegação em um documento.

A região de uma *âncora* de um nó de contexto *C* é um sub-conjunto dos nós contidos em *C*, ou o conteúdo de *C* como um todo (âncora λ).

Cada *elo* contido no *conjunto de elos* de um nó de contexto *C* define uma relação entre nós recursivamente contidos em *C* ou o próprio nó *C*, tal que todo nó cabeça do elo ou é o próprio contexto *C*, ou um nó contido (não recursivamente) em *C*. Mais especificamente, um elo define uma relação entre eventos de nós recursivamente contidos em C. Eventos serão definidos mais adiante, assim como elos NCM.

Uma *coleção de apresentação* contém para cada nó contido em um nó de contexto *C*, um grupo de conjuntos de descritores, ou o valor nulo. Um *conjunto de descritores* contém um conjunto de descritores alternativos do qual apenas um pode ser

selecionado em cada conjunto (um descritor é escolhido dependendo de parâmetros definidos na plataforma de exibição onde o documento estiver sendo exibido).

O NCM inclui a noção de estado de um nó terminal e de um nó de contexto para facilitar o controle de consistência entre nós, o suporte a trabalho cooperativo e, principalmente, a criação automática de versões. O estado é um atributo básico de um documento NCM, cujo valor afeta e é afetado pela execução de certas operações. Todo documento NCM deve possuir um método que permite alterar seu estado de forma consistente.

Um documento NCM pode estar em um dos seguintes estados: *permanente*, *temporário* e *obsoleto*. Um documento é criado no estado temporário e nesse estado permanece enquanto estiver sendo modificado. Ao tornar-se estável, o documento pode ser promovido ao estado permanente explicitamente através de uma operação do usuário ou implicitamente como efeito colateral de outras operações. Um documento permanente não pode ser diretamente removido ou alterado, mas o usuário pode torná-lo obsoleto.

OBJETOS DE ARMAZENAMENTO, DADOS E REPRESENTAÇÃO

Quaisquer das subclasses de nós mencionadas anteriormente podem ser especializadas em classes *objeto de armazenamento*, *objetos de dados* e *objetos de representação*.

Um *objeto de armazenamento - OA* representa um objeto de acesso público em um estado permanente, que de maneira geral não poderá ser alterado.

Um *objeto de dados — OD* é criado ou como um objeto totalmente novo, como uma cópia de outro objeto de dados, ou como uma cópia local de um objeto de armazenamento, acrescida de novos atributos (não-persistentes) que são dependentes da aplicação. Ele contém métodos para manipular os novos atributos, assim como métodos para manipular a informação originalmente pertencente ao objeto de armazenamento, exceto o atributo conteúdo de nós de contexto que não é interpretável nesse nível de abstração. Objetos de dados criados a partir de outros objetos são considerados *versões de dados*.

Um *objeto de representação — OR* é criado como uma cópia local de um objeto de dados, adicionando-se novos métodos para exibir e editar o atributo conteúdo (no caso de contextos a estrutura de nós e elos), no formato mais apropriado para um uso particular da informação. Um objeto de representação age como uma nova versão de um objeto de armazenamento, no caso de nós contexto e nós terminais, derivado de um objeto de dados, fato que os leva a serem considerados *versões de representação*. Objetos de representação oferecem diferentes exibições de objetos de dados, quando associados a diferentes descritores.

5.2.2.2 EVENTO

Um evento, no NCM, segue a definição de Perez-Luque [PerLi 1996], sendo definido como uma ocorrência no tempo que pode ser instantânea ou durar um período de tempo. Um evento pode representar a exibição, *evento de exibição*, ou a seleção, *evento de seleção*, de uma âncora (segmento de mídia) de um nó em uma dada perspectiva seguindo as diretrizes de um dado descritor. O NCM define também um terceiro tipo de evento, denominado *evento de atribuição*, que corresponde à mudança de um atributo de um nó em uma dada perspectiva.

Um evento pode estar em um dos seguintes estados: *dormindo*, *preparando*, *preparado*, *ocorrendo*, *suspenso* e *abortado*. A manutenção do estado desses eventos é responsabilidade da máquina de controle da apresentação dos documentos, denominada formatador, detalhada em [Rodri 1997].

5.2.2.3 ELOS

Um *elo NCM* representa um relacionamento entre eventos dos *nós NCM* que o elo referencia, podendo definir relacionamentos de causalidade ou de restrição. A definição de elos que será brevemente apresentada nesta seção, pode ser vista por completa em [MucSo 2001].

Basicamente um elo é uma entidade que tem como atributos básicos adicionais uma *relação* (conector hipermídia), e um *conjunto de binds* (ligações). O atributo *relação*

descreve o relacionamento de causalidade ou de restrição especificando papéis que serão desempenhados pelos nós interligados pelo elo, especificando como um tipo de evento participa de uma relação. O *conjunto de binds* irá conter os nós que desempenham cada papel especificado no atributo relação, determinando quais os pontos terminais de origem e destino de um elo.

Separar a especificação da relação de causalidade, ou restrição, dos nós, que efetivamente vão interagir em uma relação, trará algumas vantagens para a autoria de documentos NCM. Dentre essas vantagens, pode-se ressaltar o reuso de uma relação em elos diferentes e o reuso de uma mesma condição ou ação para vários nós que interagem através da relação.

5.2.2.4 HIPERBASE PÚBLICA E BASES PRIVADAS

A hiperbase pública corresponde a um repositório compartilhado de informação, e implementa o armazenamento persistente de objetos multimídia/hipermídia do modelo NCM. Uma base privada é utilizada para criação, edição e visualização de dados particulares de usuários.

Uma ***hiperbase pública*** H_B é um nó de composição que agrupa *nós terminais* e *nós de contexto*, documentos NCM. Todos os nós em H_B devem estar ou no estado *permanente* ou no *obsoleto*. Se um nó de contexto de usuário *C* está em H_B, então todos os nós contidos em *C* devem também estar contidos em H_B.

Uma ***base privada BP*** é também uma especialização do *nó de composição*, cujo conteúdo é formado por nós de *contexto, nós terminais*, *nós de anotação* e *bases privadas*. *BP* possui um atributo adicional, denominado *relações*, cujo conteúdo é um conjunto *R* de elos. Os elos em *BP* sempre têm como nó cabeça de seu ponto terminal de origem um nó de anotação.

Algumas restrições são feitas a bases privadas:

⇒ Uma base privada pode pertencer a no máximo uma base privada.

⇒ Se um nó de composição *N* está contido em uma base privada *BP*, seus componentes ou estão contidos em *BP*, ou na hiperbase pública, ou em qualquer base privada de um aninhamento de bases privadas onde *BP* está recursivamente contida.

⇒ Uma versão específica de um documento pode pertencer a apenas uma base, seja ela base privada ou a hiperbase pública.

A classe base privada é especializada em OD-base privada e OR-base privada.

Uma *OD-base privada ODBP* é um objeto de dados tal que:

- só contém objetos de dados;
- se um nó de composição *N* está contido na *ODBP*, seus componentes ou estão contidos em *ODBP*, ou na hiperbase pública, ou em qualquer base privada de um aninhamento de bases privadas onde *ODBP* está recursivamente contida.

Uma OR-base privada *ORBP* é um objeto de representação tal que:

- só contém objetos de representação;
- se um nó de composição *N* está contido na *ORBP*, seus componentes ou estão contidos em *ORBP*, ou em qualquer base privada de um aninhamento de bases privadas onde *ORBP* está recursivamente contida ou contidos na OD-base privada *ODBP* de onde *ORBP* foi derivada, ou em qualquer base privada de um aninhamento de bases privadas onde *ODBP* está recursivamente contida, ou contidos na hiperbase pública.

Um nó OR-base privada *ORBP* é derivado de uma OD-base privada *ODBP*. Ao nó *ORBP* serão acrescidos novos atributos e métodos para a manipulação do atributo conteúdo da base privada. Como definido, a OR-base privada conterá apenas nós de representação. Todos os objetos de dados da *ODBP* poderão ser associados a descritores, formando os objetos de representação que estarão contidos na *ORBP*. Versões de representação contidas na *ORBP* são criadas de acordo com operações de versionamento executadas, conforme serão mencionadas em 5.3.2.1. Intuitivamente, uma OR-base privada agrupa todas as entidades usadas pelo usuário durante uma sessão de trabalho.

Seja *P*" uma perspectiva para o nó *N*, objeto de dados ou de armazenamento, em uma OR-base privada *ORBP*. Seja *P*, a perspectiva correspondente a *P*" na OD-base privada correspondente a *ORBP*, contendo *N* como nó base e substituindo todos os nós de representação de *P*" pelos nós de dados de onde foram derivados. Chama-se P" a *perspectiva correlata* de P' e vice-versa.

5.2.2.5 ANOTAÇÕES

Anotações consistem de comentários (em qualquer formato ou mídia) e mantêm referências às versões, sobre as quais é feito o comentário, e às versões, consideradas respostas ao comentário.

Mais precisamente, uma *anotação A* é um nó de composição cujo conteúdo é um conjunto *S* de nós terminais e nós contextos de usuário. *A* possui um atributo adicional, cujo conteúdo é um conjunto *R* de elos, tais que todo nó cabeça de cada elo em *R* ou é o próprio contexto *A*, ou um nó de *S*.

Intuitivamente, elos de uma base privada podem ligar os componentes de um nó de anotação a nós da mesma base privada, indicando nós que originaram o comentário e nós de resposta aos comentários. Anotações irão permitir que observações sejam feitas a um nó permanente, sem a criação de novas versões.

Anotações não derivam por versionamento de nenhum nó e não podem migrar para a hiperbase pública.

5.3 MODELO DE COOPERAÇÃO SEGUNDO O NCM

O *modelo de cooperação*, apresentado no Capítulo 3, será adaptado, nesta seção, para o HyperProp, utilizando-se objetos NCM, por vezes, também devidamente adaptados, e irá permitir que, a partir da definição do *modelo de tarefas* especificado de acordo com os conceitos NCM, sejam definidos os mecanismos de colaboração, comunicação e coordenação necessários para a autoria cooperativa no HyperProp.

De maneira geral, tarefas serão representadas por *bases privadas*, e relacionadas através de *elos NCM*. A *colaboração* entre atores será realizada em *bases privadas*, a *comunicação* será auxiliada por nós de *anotações* e pelo *controle de notificação*, e a *coordenação* do trabalho, pelo *controle de acesso* apresentado no Capítulo 4, e especificado para o HyperProp no Apêndice A.

Deve-se observar que os conceitos definidos durante os Capítulos 2 e 3 serão válidos para a autoria em bases privadas, incluindo-se as definições dos mecanismos de colaboração, coordenação e comunicação, e serão muito utilizados ao longo desta seção. Quaisquer restrições ao que foi definido, serão devidamente destacadas.

5.3.1 TAREFAS X BASES PRIVADAS

Tarefas e bases privadas foram definidas basicamente com o mesmo intuito, de possibilitar a interação de usuários com documentos, na realização de um determinado trabalho e, mais especificamente, para possibilitar a autoria cooperativa de documentos hipermídia. Tarefas, assim como bases privadas, irão permitir que usuários compartilhem informação e permitir a fragmentação de um espaço de trabalho em frações menores para reduzir o escopo do trabalho.

Bases privadas, dessa maneira, irão representar tarefas, que relacionadas através de elos NCM, irão compor o *modelo de tarefas hipermídia* para o NCM. Para isso, certas características, definidas ao longo deste trabalho, deverão ser acrescentadas à definição de bases privadas, enquanto outras deverão ser adaptadas.

Uma *base privada BP*, segundo a definição de tarefas, deve definir entre outros atributos:

- um *conjunto de atores*, responsáveis pela autoria em *BP*;
- um *conjunto de documentos* contidos em *BP*, e um *conjunto de documentos contidos por recursão* em *BP*, sobre os quais a autoria em *BP* será realizada;
- um *conjunto de bases privadas*, que irão possibilitar o aninhamento de bases privadas e proporcionar a partição de um trabalho;

- um *conjunto de elos,* descrevendo o relacionamentos entre bases privadas contidas em *BP*;
- um *conjunto de anotações*, que poderão ser criadas sobre bases privadas, documentos ou anotações, contidos em *BP*;
- um *conjunto de elos de anotação,* para relacionar anotações e objetos anotados;
- o *modo de cooperação* entre os atores da tarefa, que define como notificações entre atores devem ser propagadas;
- um *controle de acesso,* que especifica regras de acesso à documentos contidos em *BP* ou à própria base privada *BP*;
- um *evento de autoria* associado.

Segundo as definições, apresentadas na Seção 5.2.2, uma *base privada BP* possui os seguintes atributos:

- *UID*: atributo original de entidade, que identifica exclusivamente *BP*;
- *Nome*: atributo original de entidade, que identifica *BP* através de uma cadeia alfa numérica (um nome não necessariamente é único no sistema);
- *Descrição*: atributo original de entidade, que representa uma descrição sobre *BP*;
- *Autor*: atributo original de entidade, que identifica os autores de *BP*;
- *Lista de descritores alternativos*: atributo original de nó, que, para bases privadas, irá conter apenas um descritor que deverá ser atribuído a *BP*;
- *Conteúdo*: atributo original de nó, que representa o conteúdo de *BP*, formado por *documentos* (*nós de contexto e nós terminais*), *nós de anotação* e outras *bases privadas*;
- *Relações*: identifica um conjunto *R* de elos que podem ligar *nós de anotação* a *documentos* contidos em *BP*.

Pode-se observar que a definição de base privada já inclui muitos dos componentes necessários e identificados no modelo de cooperação, como atributos de tarefas.

Os atributos *UID* e *nome* irão identificar uma base privada *BP*, representando, respectivamente, uma identificação única, e uma identificação alfa-numérica, através da qual *BP* poderá ser identificada informalmente em um ambiente de trabalho.

O atributo *descrição* será utilizado para representar o objetivo, ou a descrição do trabalho que deverá ser realizado em *BP*, e que deverá ser consultado durante o processo de autoria cooperativa, por indivíduos participantes desse trabalho em *BP*.

O atributo *autor* irá conter os autores responsáveis pela criação de uma base privada *BP*, que poderão ou não participar do trabalho realizado em *BP*. Nesse caso deverão estar contidos no *conjunto de atores* de *BP*.

O atributo *conjunto de atores* foi acrescentado à definição de base privada, passando a conter os *atores* que irão participar de um trabalho realizado na base privada, conforme a definição para tarefas em 2.2.1.1.

O *descritor* que será atribuído a uma base privada *BP*, contido no *conjunto de descritores* de *BP*, irá definir características de apresentação e execução para a base privada. Serão definidas, por exemplo, ações particulares a *BP*, que deverão ser executadas, de acordo com as ações definidas em elos relacionando *BP*. Essas ações serão àquelas definidas em 2.2.3, para tarefas. Descritores não tinham sido definidos no *modelo de cooperação*, apresentado em 3.4, mas serão utilizados aqui como meio de separar as características de apresentação e execução de tarefas, assim como ocorre para documentos NCM.

A definição do atributo *relações* foi estendida para permitir a definição de elos relacionando *bases privadas*, *nós de anotação* e *bases privadas*, e *nós de anotação* e *nós de anotação*, já que esse atributo previa apenas o relacionamento entre *nós de anotação* e *documentos*. Diz-se que um elo *L* está contido em *BP*, quando pertencente ao conjunto de elos, definido pelo atributo *relações*, de uma base privada *BP*. Um elo, contido em uma base privada *BP*, tem como ponto terminal de origem um *nó de anotação* ou uma

base privada, ambos contidos[30] em *BP*. No primeiro caso, o elo será chamado de *elo de anotação*, e poderá ter como ponto terminal de destino um *nó de anotação*, um nó *base privada*, ou *documentos*, já que anotações poderão ser feitas sobre *anotações*, *bases privadas* ou *documentos*, como apresentado em 3.2.1 . No segundo caso, quando o ponto terminal de origem do elo é uma *base privada*, seu ponto terminal de destino deverá ser necessariamente outra *base privada*, contida em *BP*.

Como foi dito anteriormente, o conteúdo de uma base privada será composto por bases privadas, documentos e nós de anotação. Como foi definido, no entanto, o atributo (ou propriedade) *conteúdo* irá restringir o acesso a bases privadas, documentos, e nós de anotação como componentes de um único conjunto representado por esse atributo. Muitas vezes, no entanto, durante a autoria cooperativa, pode-se desejar que regras de acesso distintas sejam determinadas para cada um dos subconjuntos implicitamente existentes: o conjunto de bases privadas, o conjunto de documentos, e o conjunto de nós de anotação. A possibilidade de se definir direitos de acesso diferentes sobre esses conjuntos, irá permitir, por exemplo, a distinção entre atores com permissão para alterar a estrutura de uma determinada base privada *BP* (inserindo e removendo bases privadas do conjunto de bases privadas de *BP*), e aqueles com permissão para atuar durante o trabalho em *BP* (inserindo e removendo documentos do conjunto de documentos de *BP*), mas que não poderão alterar a estrutura de *BP*.

Nesse sentido, para que fosse possível o tratamento diferenciado desses conjuntos de *documentos*, *bases privadas* e *nós de anotação*, contidos em uma base privada, optou-se por alterar a definição de *base privada* e *conteúdo*, de maneira que, a propriedade *conteúdo* passou a conter apenas o conjunto de documentos contidos na base privada, enquanto que foram criadas as novas propriedades, *conjunto de bases privadas* e *conjunto de nós de anotação*, que passaram a pertencer ao conjunto de propriedades de uma entidade base privada.

---

[30] A definição de contingência de um nó de anotação, base privada ou documento, segue as mesmas definições para tarefa, e poderá ser vista em seguida.

O novo atributo *conjunto de bases privadas* de uma base privada *BP* irá conter as bases privadas antes contidas no *conteúdo* de *BP*. Diz-se que uma base privada *BP'* está contida em *BP*, se está contida no *conjunto de bases privadas* de *BP*[31]. O novo atributo *conjunto de anotações* de uma base privada *BP* irá conter nós de anotações antes contidas no *conteúdo* de *BP*, representando anotações, como foi definido para tarefas em 3.2.1. Diz-se que um *nó de anotação* N_a está contido em *BP*, se está contido no *conjunto de anotações* de *BP*.

Ainda, para que documentos contidos em uma base privada, na qual *BP* está recursivamente contida (documentos visíveis a *BP* e aos atores de *BP*, como foi definido para tarefas em 3.1.1), pudessem ser alterados em *BP*, foi acrescido ao conjunto de propriedades de uma base privada, o atributo *conjunto de documentos contidos por recursão* (C_{dcr}), como o foi para tarefa. Esse conjunto será representado em *BP* por um nó de contexto N_{dcr}, chamado de *contexto de documentos contidos por recursão*. O conteúdo de N_{dcr} será formado pelos nós contidos em C_{dcr}, e, apesar de ser um nó de contexto, não poderão ser criados elos em N_{dcr}, já que esse contexto irá representar uma base de documentos (como bases privadas e a hiperbase pública), onde elos entre documentos não podem ser criados diretamente na base. Deve-se notar que N_{dcr} foi criado apenas para que um nó *N* visível a *BP* pudesse ser alterado em *BP*, atendendo-se a restrições de que um documento só pode estar em uma única base de trabalho.

Foram acrescentados, ainda, ao modelo NCM, como novos atributos de base privada, o *evento de autoria*, e um *controle de acesso*, definidos para tarefas.

O *evento de autoria* foi incluído no modelo para representar o evento ocorrido em uma base privada *BP*, durante a realização de um trabalho em *BP*, assim como foi apresentado na Seção 2.2.2 para tarefas. Os possíveis estados de um *evento de autoria* representarão os estados possíveis de uma tarefa, representada por uma base privada, e servirão para retratar as diferentes fases da autoria cooperativa.

---

[31] A noção de contingência por recursão é a mesma que aquela definida anteriormente para nó de composição e para tarefas.

O *controle de acesso* foi incluído para possibilitar a definição de regras de acesso à base privada e a seus atributos, e a documentos contidos na base privada, como parte do mecanismo de coordenação do trabalho realizado em bases privadas, assim como o foi para tarefas.

Uma base privada, ainda, além de definir os atributos apresentados anteriormente, deve definir métodos para manipular esses atributos, como os métodos para inserir atores e remover atores, inserir nós (bases privadas, documentos e anotações), inserir elos (elos entre bases privadas e elos de anotação), remover nós e remover elos, e métodos para exibir seu conteúdo[32] (bases privadas, documentos contidos e contidos por recursão, e de nós de anotação).

ELOS ENTRE BASES PRIVADAS

Elos NCM serão utilizados diretamente para especificar as relações entre bases privadas.

Elos NCM, definidos entre *bases privadas*, irão conter especificações de seus relacionamentos e definir condições e ações (no caso de elos de causalidade), e restrições (no caso de elos de restrição) a partir de alterações de estado em *eventos de autoria* relativos a *bases privadas*. A definição de ações em *elos* entre bases deverá ser compatível com a definição de relacionamentos em 2.2.3.

Exemplos de definição de elos poderão ser vistos em 5.4.2.

5.3.2 COOPERAÇÃO EM BASES PRIVADAS

A autoria cooperativa no HyperProp será realizada através de colaborações coordenadas entre atores em uma *base privada*, utilizando *nós de anotação*, como instrumento de comunicação, e o Controle de Acesso, definido no Capítulo 4 e especificado de acordo

[32] Entende-se por conteúdo aqui, os componentes que estão contidos em uma base privada, o que não está limitado aos componentes do atributo *conteúdo* de uma base privada.

com as necessidades do NCM, no Apêndice A, como mecanismo de coordenação do trabalho.

Ainda, serão utilizados, os controles de versão e notificação, definidos em [Soare 1999] e [Muniz 2000] para o sistema HyperProp, como mecanismos de suporte à cooperação.

5.3.2.1 COLABORAÇÃO

A **colaboração** entre atores será realizada em bases privadas, propriamente estruturadas, como foi apresentado na Seção 3.1.

A autoria de documentos em bases privadas poderá ser realizada sobre documentos versionáveis ou não versionáveis, cabendo ao *controle de versão* definido para o NCM, a manipulação de diferentes versões de documentos. Mais especificamente, atributos de documentos poderão ser definidos como versionáveis ou não. Atributos não versionáveis poderão ser modificados sem que uma nova versão de um documento tenha que ser criada, enquanto que a alteração em atributos versionáveis só será possível em uma nova versão de um documento. Dessa maneira, a definição de um atributo como versionável ou não irá determinar o uso ou não de versões durante a autoria cooperativa de documentos.

Em ambos os casos, a restrição que se faz à autoria cooperativa de documentos NCM em bases privadas é aquela já definida no modelo: se um determinado documento está contido em uma base privada *BP*, seus componentes devem estar contidos em *BP*, em qualquer base privada de um aninhamento de bases privadas onde *BP* está recursivamente contida, ou em uma hiperbase pública[33]. Ainda, uma versão de um documento deve pertencer a uma única base de trabalho, incluindo-se aí bases privadas e hiperbase pública. É importante ressaltar que a autoria em documentos deverá ser realizada de acordo com as definições e restrições do modelo conceitual de dados NCM.

Segundo o modelo, uma versão de um documento poderá estar nos estados temporário, permanente ou obsoleto, que servirão para retratar as diferentes fases da autoria de documentos e possibilitar a manipulação de versões pelo *controle de versões*. Esses estados dizem respeito aos atributos versionáveis de um documento, que não podem ser modificados diretamente, quando um documento está em um estado permanente, ou que podem ser modificados, quando um documento está em um estado temporário. Regras para alteração em atributos não versionáveis devem ser estabelecidas através de um controle de acesso.

Durante a autoria cooperativa, documentos poderão se "movimentar" entre bases de trabalho, permitindo que sejam utilizados e expostos em diferentes *ambientes de trabalho*, conforme determinadas regras e de acordo com as necessidades do trabalho.

Um documento *D* contido em uma determinada base privada *BP*, não poderá migrar para outra base privada *BP'*, recursivamente contida em *BP*, embora possam ser criadas diferentes versões desse documento[34] em *BP'*. Essa restrição irá evitar que inconsistências sejam criadas no que diz respeito às definições de visibilidade e uso de documentos em bases privadas aninhadas. Imagine, por exemplo, que um determinado documento *D*, contido em uma base privada *BP1*, tenha como um de seus componentes um outro documento *C* contido em uma base privada *BP*, na qual *BP1* está recursivamente contida. Se *C* fosse movido para uma base privada *BP'* recursivamente contida em *BP*, mas na qual *BP1* não estivesse recursivamente, uma inconsistência seria criada.

Da mesma forma que para bases privadas, um documento *D* contido em uma hiperbase pública não poderá migrar para uma base privada *BP*, embora possam ser criadas versões de *D* em *BP*.

---

[33] Essa restrição é um pouco diferente dependendo se o documento é um documento objeto de representação ou dados, como foi mostrado em 5.2.2.4.

[34] O documento deve estar em um estado permanente.

Conforme o que foi apresentado em 3.1.2, documentos, contidos em uma base privada *BP*, poderão migrar para outra base privada *BP'*, na qual *BP* está contida, para que o resultado de um trabalho realizado (documentos no estado permanente) em *BP* possa ser visível a outras bases, além daquelas recursivamente contidas em *BP*. Nesse caso, apenas documentos no estado permanente, e, portanto consistentes, poderão ser transferidos. Note que nesse caso, inconsistências não serão criadas e a restrição de visibilidade sempre será garantida, já que documentos sempre serão movidos para uma base mais externa, visível às bases privadas recursivamente contidas.

Para possibilitar esse "movimento" de documentos entre bases privadas, foram utilizadas as primitivas *check-out (P, C_{ont})*, para criar versões de documentos contidos em uma base privada *BP*, em outra base privada recursivamente contida em *BP*, e *move (D)*, para que um documento *D*, contido em *BP*, seja transferido para uma base privada na qual *BP* está contida.

A figura seguinte ilustra como o movimento de documentos ocorre entre bases de trabalho.

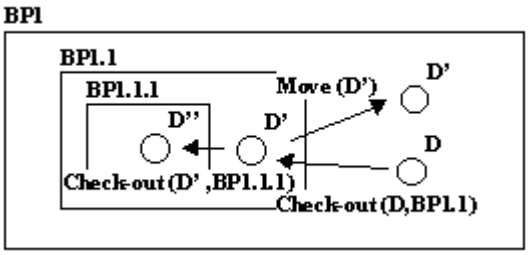


Figura 5.2: Primitivas para movimento de documentos em bases privadas.

A primitiva *check-out (P, C_{ont})* permite a criação de uma nova versão de um documento *D*, nó base da perspectiva *P*, contido em uma base privada *BP* em uma base privada *BP'*, recursivamente contida em *BP*. Se já existir uma versão do documento derivada de *D* em

BP', recursivamente contida no contexto C_{ont}, a nova versão não é criada[35]. C_{ont} pode receber o valor nulo (f), indicando que a versão deverá obrigatoriamente ser criada.

Nesse caso, uma *perspectiva P*, deve definir uma seqüência P = (N_m,....,N_1), com $m > 1$, tal que $N_1 = D$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para i ? [1,m), e N_m é a base privada aonde a versão de *D* será criada. Deve-se notar que se um determinado nó N_i é um nó base privada, uma seqüência $P' = (N_m,....,N_i)$, com m>i>1, é uma seqüência que, obrigatoriamente, contém apenas nós base privada. Por outro lado, se um nó N_i é um nó de contexto, a seqüência $P'' = (N_i,....,N_2)$, com i>3, é uma seqüência que, obrigatoriamente, contém apenas nós de contexto.

Na figura, quando uma operação *check-out (BP1.1, D, BP1.1)* é realizada em *BP1.1*, onde P = *BP1.1*, *D*, e C_{ont} = *BP1.1*, uma versão do nó *D* é criada em *BP1.1*, já que não existem versões de *D* no contexto C_{ont}

A primitiva *move (D)* permite a transferência de um documento *D* no estado permanente de uma base privada *BP'* para uma base privada *BP*, na qual *BP'* está contida, para que o trabalho seja exposto em *BP* e seja visível a *BP* e às bases privadas recursivamente contidas em *BP*. Quando *D* for movido para *BP*, todos os componentes contidos em *D* também serão transferidos para a mesma base *BP*.

Na figura, quando uma operação *move (D)* é realizada em *BP1.1*, o documento *D* e seus componentes são movidos para a base privada *BP1*. Deve-se notar que *D* será removido do conjunto de documentos (*conteúdo*) de *BP1.1* e inserido no conjunto de documentos (*conteúdo*) de *BP1*.

AUTORIA DE DOCUMENTOS EM PLANOS DE TRABALHO

A autoria de documentos no NCM será realizada sobre documentos objetos de representação, criados ou não como versões de documentos objetos de dados, onde

[35] C_{ont} pode ser utilizado para que se evite a criação de versões diferentes de um mesmo documento em uma mesma base privada.

objetos de dados podem ser associados a diferentes descritores para formar objetos de representação distintos. Versões de representação estarão contidas em OR-bases privadas, enquanto versões de dados estão contidas em OD-bases privadas.

Um nó em uma OR-base privada poderá ser criado como um novo nó ou como uma versão de um nó contido em uma OD-base privada correlata. Um nó em uma OD-base privada *ODPB* poderá ser criado como um novo nó, como uma versão de um nó contido na *ODBP*, de um nó contido em uma *ODBP'* na qual *ODBP* está recursivamente contida, ou como uma versão de um nó na hiperbase pública. Um nó na hiperbase pública não pode migrar desta hiperbase para uma OD-base privada, embora possam ser criadas versões de dados deste nó nas diversas OD-bases privadas. Analogamente, um nó em uma OD-base privada não pode migrar desta base para uma OR-base privada, embora possam ser criadas versões de representação deste nó na OR-base privada. Versões de representação poderão ser derivadas a partir de um nó objeto de dados permanente ou temporário.

A primitiva *check-out* (P, C_{ont}), como foi definida anteriormente, permite a criação de uma nova versão de dados de um nó N contido na hiperbase pública, ou contido em uma OD-base privada, levando-se em conta a definição de visibilidade de documentos[36].

A primitiva *check-out* (P, D_e, C_{ont}) permite a criação de uma nova versão de representação de um objeto de dados D, onde D (no estado permanente ou temporário) é o nó base da perspectiva P[37]. Todos os nós de P devem ser nós objeto de representação, exceto o nó D. A versão, que deve ser incluída na OR-base privada *ORBP*, é criada pela associação do nó D ao descritor D_e especificado. Se já existir uma versão de representação de D em *ORBP*, derivada do mesmo descritor D_e, e recursivamente contida no contexto C_{ont}, a nova versão não é criada. C_{ont} pode receber o valor nulo (f), obrigando que uma nova versão seja sempre criada.

[36] Versões de documentos, contidas em uma base privada *ODBP*, poderão originar novas versões apenas em OD-bases privadas recursivamente contidas em *ODBP*.

Quando a primitiva *check-out (P, D_e, C_{ont})* for aplicada, uma nova versão de *D*, nó base da perspectiva *P*, deverá ser criada na OR-base privada *ORBP*, correlata à OD-base privada aonde *D* está contido, e dependerá da perspectiva *P* a ação que deverá ser tomada posteriormente. Nesse sentido, deve-se considerar as seguintes situações:

⇒ A perspectiva *P* define apenas uma base privada - *ORBP*.

- Se *P* define um nó de contexto *C* (contido ou não em um aninhamento de nós de contexto), que deve conter a versão de representação do nó *D* em *ORBP*, uma versão de *D* ainda é inserida no contexto *C*.

⇒ A perspectiva *P* define um aninhamento de OR-bases privadas recursivamente contidas em *ORBP*. Nesse caso, suponha que o aninhamento de bases privadas é dado pela perspectiva $B = (B_m,....,B_1)$, com $m > 1$, tal que *B* está contido em *P*, e $B_m = ORBP$.

- Se *P* define um nó de contexto *C* (contido ou não em um aninhamento de nós de contexto), que deve conter a versão de representação do nó *D*, uma versão de *D* ainda é inserida no contexto *C* contido em B_1.

- Caso contrário, a versão não poderá ser inserida diretamente em B_1, já que uma determinada versão só pode estar contida em uma única base (base privada ou hiperbase pública), e essa já está contida em *ORBP*. Como solução a esse problema, a versão, nesse caso, será inserida no *conjunto de documentos contidos por recursão* da base B_1, e automaticamente no *contexto de documentos contidos por recursão* N_{dcr},. A versão de *D*, dessa maneira, poderá ser manipulada e alterada em B_1.

É importante notar que outras observações a respeito da criação de versões (de dados ou representação), através das primitivas apresentadas, são feitas em [Soare 2000].

As figuras seguintes ilustram algumas situações ocorridas durante a autoria de documentos em OR-bases privadas de forma a ilustrar as primitivas, apresentadas anteriormente.

---

[37] A definição da perspectiva *P*, para a primitiva *check-out (P, C_{ont})* também é válida aqui.

A Figura 5.3 retrata a autoria em uma versão de um documento, na base privada *BP1.1'*, de maneira tal que esse trabalho seja visível a apenas *BP1.1'* (e às possíveis OR-bases privadas recursivamente contidas em *BP1.1'*). Ao final do trabalho, no entanto, a versão deverá ser transferida para uma base privada mais externa, para que possa ser visível a *BP1'* e às bases privadas recursivamente contidas em *BP1'*.

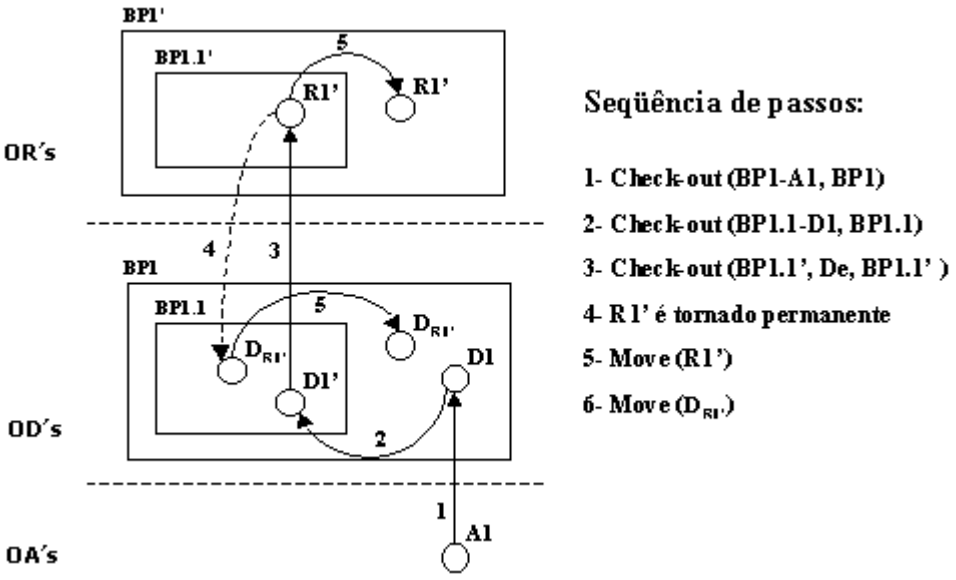


Figura 5.3: Autoria em *ORBP* – Exemplo 1.

A autoria, em *BP1.1'*, será realizada sobre uma versão de um documento *D1*, contido na base privada *BP1*, derivada, inicialmente, do documento *A1*, contido na hiperbase pública (passo 1). Para que seja criada uma versão de representação desse documento na base desejada (*BP1.1'*), fornecendo a visibilidade desejada, dois passos devem ser dados: Deve ser criada uma versão de *D1* em *BP1.1* (*D1'*), que será visível somente à OD-base privada *BP1.1*, às OD-bases privadas recursivamente contidas em *BP1.1*, e às OR-bases privadas relacionadas (passo 2); e, a partir de *D1'*, uma versão de representação *R1'* em *BP1.1'* deve ser criada, como desejado (passo 3).

Nesse caso, pode-se definir uma operação composta que realize esses dois passos automaticamente. Seja *D* um objeto de dados terminal ou de contexto, onde *D* é o nó base da perspectiva *P''* em uma OR-base privada *ORBP*. Seja *P'*, a perspectiva correlata de *P''* na OD-base privada *ODBP* correspondente a *ORBP*. A primitiva *Check-out (P'',D_e)* irá permitir a criação de uma nova versão de representação de *D*, primeiramente realizando

a operação *Check-out (P', f)*, em *ODBP*. No exemplo da figura, a operação composta seria definida da seguinte maneira: *Check-out (BP1.1',D1, D_e)*, quando seriam executadas as primitivas *Check-out (BP1.1,D1, f)* e, em seguida, *Check-out (BP1.1',D1', D_e, f)*.

Finalizado o processo de autoria em *BP1.1'*, a versão *R1'* poderá ser transferida para a base privada *BP1'*, depois que estiver em um estado permanente[38] (passo 4). De maneira geral, tornar um documento objeto de representação *DR* permanente, significa tornar as alterações realizadas na versão de representação *DR* permanentes em um objeto de dados correspondente. Suponha-se que, no exemplo, tornar *R1'* permanente significou a criação do objeto de dado $D_{R1'}$ em *BP1.1*, versão do objeto *D1*, para refletir as alterações realizadas na versão *R1'*, criada a partir da associação do objeto de dados *D1'* com o descritor D_e.

Depois de tornado permanente, o documento *R1'* pode ser movido para *BP1'*, através da primitiva *move (R1')* (passo 5), assim como o objeto de dados equivalente pode ser movido para *BP1*, através da primitiva *move ($D_{R1'}$)* (passo 6). A transferência do objeto $D_{R1'}$, correspondente a *R1'*, para *BP1*, irá possibilitar a criação de novas versões que contenham as alterações realizadas em *R1'*, em *BP1* ou nas OD-bases privadas recursivamente contidas em *BP1*, ou em *BP1'* ou nas OR-bases privadas recursivamente contidas em *BP1'*.

Nesse caso, uma operação composta também poderia ser definida para que as duas operações pudessem ser realizadas em seqüência. Seja *R* um objeto de representação terminal ou de contexto em uma OR-base privada *ORBP*, e *D* o objeto relacionado a *R* na OD-base privada *ODBP* correspondente a *ORBP*. A operação *Transfer (R)* executada em *ORBP* irá executar a operação *Move (R)* em *ORBP*, e logo após a operação *Move (D)* em *ODBP*.

[38] Tornar um documento permanente traz uma série de conseqüências para o controle de versões, que poderão ser observadas em [Soare 2000].

A Figura 5.4 retrata o caso em que durante a autoria, em uma determinada base privada, sobre um documento *D* qualquer, deseja-se realizar um acesso a um documento contido em *D* que não está contido na base privada.

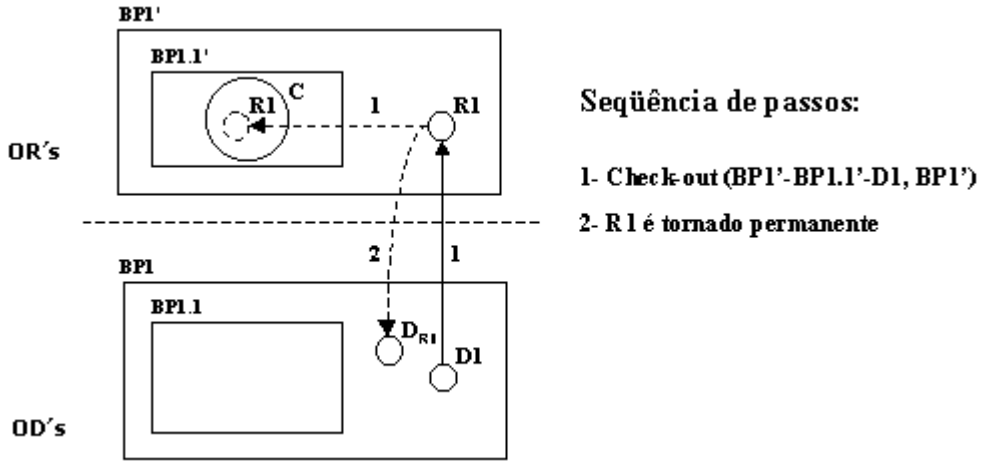


Figura 5.4: Autoria em *ORBP* – Exemplo 2.

No exemplo, em *BP1.1'*, através de uma navegação no contexto *C*, contido na base privada, deseja-se acessar um nó *D1*, componente de *C*, que está contido na OD-base privada *BP1*. Nesse caso, uma versão de *R1* será criada na OR-base privada relacionada à OD-base privada onde o nó *D1* está contido, e inserido no contexto *C* (passo 1). Quando *R1* for tornada permanente, as alterações realizadas nessa versão deverão ser refletidas em uma versão de dados na base correlata a *BP1'*, onde a versão foi criada (passo 2). Nesse caso, a versão D_{R1} foi criada para representar essas alterações em *BP1*.

Por último, deve-se observar que quando um trabalho for concluído por completo, os documentos criados e alterados em um estado permanente poderão ser movidos para a hiperbase pública e disponibilizados para outros atores que não àqueles pertencentes às bases privadas que compõem a estrutura criada para a realização de um determinado trabalho. Isso deverá ser feito através de uma primitiva de *check-in*, como já foi definido em [Soare 2000]. De maneira geral, a primitiva foi definida para objetos de representação e dados com o objetivo de mover objetos do plano de objetos de representação para o plano de objetos de dados e do plano de objetos de dados para o plano de objetos de

armazenamento[39], respectivamente, onde apenas objetos no estado permanente poderão sofrer um *check-in*.

5.3.2.2 COMUNICAÇÃO

Como dito anteriormente, a comunicação irá possibilitar a troca de informações durante o processo cooperativo, e representará um fator fundamental para a interação do grupo de trabalho. Nesse sentido, as definições de uma estrutura de bases privadas para realização do trabalho, e de primitivas que deverão possibilitar a troca de documentos entre bases de trabalho forneceram o primeiro passo para viabilizar a comunicação entre atores participantes de uma tarefa.

Nós de anotação irão possibilitar uma forma de comunicação mais explícita, permitindo a definição de anotações sobre documentos, sobre bases privadas, ou mesmo sobre outros nós de anotações. Notificações também irão exercer um papel importante durante a comunicação entre atores engajados em processo cooperativo.

NÓS DE ANOTAÇÃO

Nós de anotação poderão ser utilizados como veículo de comunicação entre os participantes de um trabalho, hora para possibilitar o diálogo entre atores, hora como sugestão sobre trabalhos, hora como respostas a outros nós de anotação. A funcionalidade de um *nó de anotação* poderá ser descrita através do atributo *descrição*, definido para qualquer *entidade NCM.*

Nós de anotação, contidos em uma base privada *BP*, serão visíveis a *BP* e às bases privadas recursivamente contidas em *BP*, como foi definido para tarefas em 3.2.1.

Um *nó de anotação* N_a contido em *BP* poderá representar uma anotação sobre documentos visíveis a *BP*, sobre bases privadas contidas em *BP*, ou sobre outros *nós de*

[39] Intuitivamente, os planos de objetos de representação, dados e armazenamento irão representar os panos onde estão contidos objetos de representação, dados e armazenamento.

anotação também contidos em *BP*, que serão relacionados a N_a através de um *elo de anotação*.

Um *elo de anotação* é um elo NCM que relaciona um *nó de anotação*, e está contido em *BP* quando relaciona um *nó de anotação* contido em *BP*. Diz-se que um *elo de anotação* está contido em uma base privada *BP*, se está contido no conjunto de *relações* de *BP*.

Nós de anotação NCM não poderão ser modificados, mas poderão ser tornados *obsoletos*, quando a anotação se tornar obsoleta. Caso contrário, estarão em um estado *ativo*. Nesse sentido, será acrescentado, à definição de *nó de anotação*, um atributo *estado* e, conseqüentemente, todo nó de anotação deve possuir um método para alterar o seu estado de maneira consistente.

O uso de nós de anotação possibilitará o armazenamento do histórico de alterações e tomada de decisões durante a realização de tarefas, além de possibilitar a comunicação assíncrona entre atores. *Nós de anotação*, a princípio, são definidos como veículo de comunicação durante apenas a execução do trabalho, em bases privadas.

NOTIFICAÇÃO

O controle de notificação, definido e apresentado em [Muniz 2000], constitui o primeiro passo para que, efetivamente, se possa fornecer um mecanismo de propagação de notificações que atenda às necessidades de um processo de autoria cooperativa de documentos em bases de trabalho no HyperProp.

A forma como o controle de notificação foi especificado, no entanto, irá possibilitar, de maneira relativamente simples, a adição de novas regras, ao conjunto de regras já definido, para o envio de notificações, como poderá ser visto em seguida.

A especificação do controle de notificação em [Muniz 2000] foi baseada no paradigma *Publish/Subscribe* [MHJPR 1995]. A idéia essencial desse paradigma é que um ou mais editores (*publishers*) enviam dados para múltiplos assinantes (*subscribers*), em

determinados canais onde assinantes se inscrevem para receber mensagens publicadas por editores.

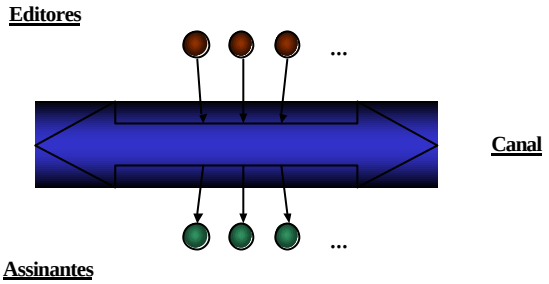


Figura 5.5: ***Publish/Subscribe***

Como foi definido o controle de notificação para o HyperProp, nós NCM representarão editores e assinantes, e canais, determinados eventos, ocorridos em nós NCM. O controle permite a inserção e remoção de assinantes e editores em canais, assim como a inserção ou remoção de canais.

Já em [Muniz 2000], foram definidos alguns canais para representar os seguintes eventos: criação de versão, atualização de atributos, obsolescência de nós, remoção e adição de elos, remoção e adição de nós. Ainda foram definidas determinadas regras de inscrição nesses canais, o que facilitou em muito o processo de inscrição em canais, nos casos em que já se sabe que determinadas inscrições são necessárias.

Através do controle de notificação, uma versão *V'*, derivada de um nó *V*, poderia se inscrever para receber notificações sempre que *V* sofrer uma atualização em qualquer de seus atributos não versionáveis, do mesmo jeito que uma base privada poderia se inscrever para receber notificações sobre alterações em atributos não versionáveis de um determinado nó, ou nós.

Os canais já definidos em [Muniz 2000], todavia, ainda não são suficientes para fornecer a consciência necessária para um grupo de trabalho sobre eventos ocorridos durante a autoria cooperativa em tarefas. Nesse sentido, pode-se citar a necessidade de notificações sobre o bloqueio e desbloqueio de um atributo e sobre alteração de estado de um evento de autoria.

A decisão sobre quais eventos deverão gerar notificações durante um trabalho cooperativo, assim como a definição e inserção de novos canais no controle de notificação, ficará como trabalho futuro. Definidos esses canais, ainda, poderão ser criadas regras de inscrição automáticas, de acordo com as necessidades de um determinado trabalho realizado em uma base privada, que, por exemplo, poderiam ser criadas de acordo com o *modo de cooperação* definido para cada base privada.

5.3.2.3 COORDENAÇÃO

A estrutura de bases privadas, elaborada para a realização de um trabalho cooperativo, e a definição de permissões de acesso a atores sobre bases privadas e documentos, através de um Controle de Acesso especificado para o HyperProp, durante um trabalho cooperativo em uma determinada base privada, irão caracterizar os mecanismos de coordenação especificados para o HyperProp.

O Controle de Acesso, que será atribuído à cada *base privada*, é uma especialização daquele apresentado no Capítulo 4, e possibilitará a definição de regras de acesso sobre entidades e propriedades NCM, como pode ser visto no Apêndice A, possibilitando a cooperação a nível de atributos de objetos NCM.

A definição de direitos de acesso, assim como a definição de travas, por atributos (propriedades) de objetos NCM foi possível graças ao *modelo de objetos* apresentado em 5.2.1, que permite a definição de entidades como uma composição de atributos, representados por objetos *propriedades*.

Como foi definido em 3.3, o Controle de Acesso, especificado para o NCM, ainda poderá ser especializado em um *Controle de Acesso para Tarefas* e um *Controle de Acesso para Documentos*, quando entidades forem especializadas em bases privadas e documentos (nós de contexto ou nós terminais), respectivamente. Essas especializações, para o HyperProp, irão determinar as pemissões relevantes para bases privadas e documentos, respectivamente, além de armazenar informações sobre o controle de travas.

Ainda, deve-se lembrar que o estado do *evento de autoria* relativo a uma base privada *BP* deverá ser levado em conta durante a realização do trabalho em *BP*, de maneira a atender às restrições apresentadas nas Seções 2.2.2 e 3.1.2.

⇒ Quando o *evento de autoria* relativo a *BP* se encontra em um estado *preparado, suspenso ou dormindo,* alterações em documentos em *BP* não poderão ser realizadas, assim como alterações no atributo *conteúdo* de *BP*. Alterações em outros atributos de *BP* poderão ser realizadas.

⇒ Quando o *evento de autoria* relativo a uma base privada *BP*, se encontra no estado *esperando conclusão*, ou *concluído*, ou *obsoleto*, alterações em atributos de *BP* não poderão ser realizadas, exceto no estado *esperando conclusão*, quando anotações poderão ser inseridas e removidas do atributo *conjunto de anotações* de *BP*.

⇒ Quando o *evento de autoria* de *BP* está nos estados *ocorrendo sem subtarefas*, *ocorrendo com nenhuma subtarefa concluída*, *ocorrendo com pelo menos uma subtarefa concluída e pelo menos uma não concluída* ou *ocorrendo com todas as subtarefas concluídas*, o acesso a documentos em *BP*, ou aos atributos de *BP*, a princípio é permitido, sendo controlado pelo controles de acesso e versão.

Por último, vale observar que, a definição de papéis, apresentada em 2.2.1.1, como meio de coordenação do trabalho em tarefas, pode ser vista como um facilitador no processo de definição de direitos de acesso a atores em bases privadas sobre suas propriedades. Dessa maneira, atores com papel de *coordenador* ou *resolvedor*, papéis esses definidos em 2.2.1.1, pertencentes a uma base privada *BP*, poderiam ser vistos como atores com determinadas permissões preestabelecidas sobre propriedades de *BP*. Um ator *coordenador* teria controle completo (uma permissão de *full control*) sobre as propriedades de *BP*, enquanto um ator *resolvedor* possuiria permissão de escrita (*Write*) sobre os atributos *conjunto de anotações* e *relações*, e permissão de leitura (*Read*) sobre os demais.

5.3.3 ESTRUTURA DO MODELO DE COOPERAÇÃO

Nesta seção, será apresentada a estrutura do *modelo de cooperação*, que dará suporte à definição de mecanismos que efetivamente irão possibilitar a autoria cooperativa de documentos no HyperProp. Essa estrutura é uma adaptação daquela apresentada em 3.4.1, e foi definida utilizando-se objetos do modelo conceitual NCM, de acordo com o que foi apresentado neste capítulo.

O *modelo de cooperação* próprio ao NCM irá relacionar os componentes definidos ao longo deste capítulo como atributos de base privada, componente básico do modelo, como pode ser visto na figura seguinte, através de um diagrama de classes em UML.

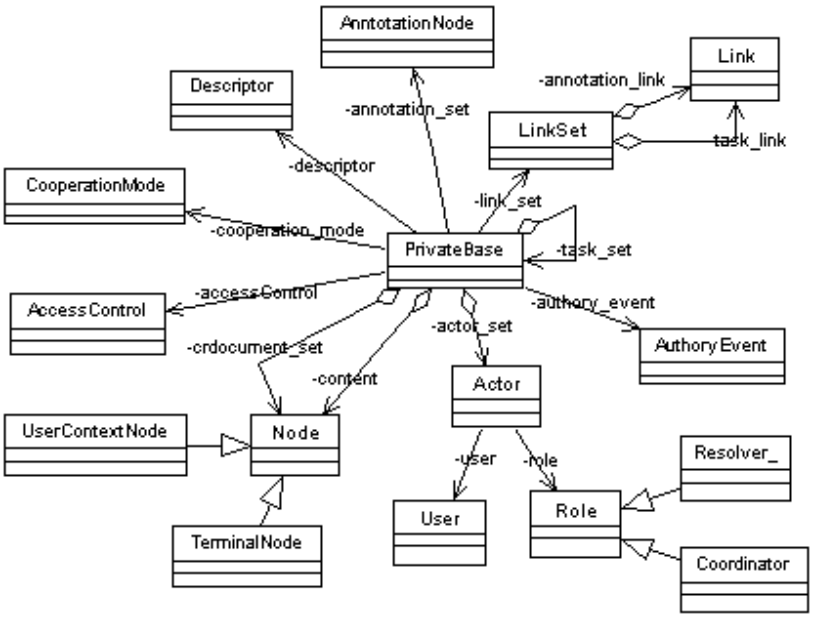


Figura 5.6: Modelo de cooperação segundo o NCM.

A entidade *base privada* tem como atributos básicos: um *conjunto de bases privadas* (*task_set*); um *conjunto de atores* (*actor_set*); o *modo de cooperação*, segundo o qual atores irão cooperar (*cooperation_mode*); o *evento de autoria* (*authory_event*); um *descritor* (*descriptor*); o atributo *conteúdo* (*content*), que irá conter o conjunto de documentos contidos na base privada; o atributo *conjunto de documentos contidos por recursão* que contém os documentos utilizados, mas que não estão contidos, na base privada (*crdocument_set*); um *conjunto de nós de anotação* (*annotation_set*); o atributo

relações (*link_set*), que contém elos relacionando *bases privadas* (*task_link*) e *nós de anotação* (*annotation_link*); e, um controle de acesso (*accessControl*) associado.

Cada uma das classes na figura representa os componentes do modelo, que devem ser especificados de modo a descrever as características apresentadas para cada um ao longo deste capítulo. Nesse sentido, alguns comentários devem ser feitos.

O *conteúdo* (*content*) de uma base privada *BP* foi especificado de modo a conter os documentos utilizados durante o trabalho e contidos em *BP*. Como foi dito, para possibilitar a autoria em documentos visíveis à *BP*, foi criado o *conjunto de documentos contidos por recursão* (C_{dcr}).

Como foi dito anteriormente, um ator (*Actor*) irá descrever um papel (*role*) que deve ser atribuído a um usuário (*user*), durante a realização de um trabalho em uma base privada *BP*, onde papéis irão agir como facilitadores durante a definição de direitos de acesso a atores de *BP*, atuando junto ao Controle de Acesso (*accessControl*) definido para essa base. No caso, dois papéis foram especificados (*coordinator* e *resolver*), como o foram na Seção 2.2.

A classe *AuthoryEvent*, representa o evento de autoria relativo a uma base privada, e é nela onde serão definidos os possíveis estados do *evento de autoria*, de acordo com a máquina de estados definida em 2.2.2. Na classe *Descriptor*, ações para a apresentação e execução de uma determinada base privada, são definidas de acordo com as ações apresentadas em 2.2.3, e de acordo com o estado do evento de autoria.

O diagrama para o controle de acesso é idêntico àquele apresentado na Figura 3.7.

5.4 AMBIENTE DE COOPERAÇÃO PARA O HYPERPROP

Esta seção apresenta um ambiente que dará suporte a definição de determinados conceitos apresentados ao longo deste capítulo. O objetivo desse ambiente é fornecer o suporte inicial necessário à definição dos objetos NCM que irão compor e participar do processo de autoria cooperativa de documentos.

Esse ambiente foi implementado em Java, como uma extensão de um ambiente gráfico para a autoria e apresentação de estruturas de documentos NCM em bases privadas, especificado em [Pinto 2000]. O ambiente de cooperação irá oferecer uma interface para a visualização e definição de bases privadas, de relacionamentos entre bases privadas, de documentos (nós de contexto e nós terminais), anotações (nós de anotação), e elos de anotação, graficamente, como foi possível para a definição de documentos em bases privadas (não aninhadas) em [Pinto 2000].

O ambiente foi implementado através de um *browser de tarefas*, onde bases privadas poderão ser estruturadas e definidas, para representar um determinado trabalho. Bases privadas, nesse ambiente, irão representar tarefas, e por isso serão chamadas de tarefas durante esta seção.

5.4.1 O BROWSER DE TAREFA

Basicamente, o *browser* é composto por uma janela (*TreeGraphBrowserWindow*) onde será apresentada uma visão da hierarquia das tarefas que irão compor um determinado trabalho, em uma árvore de tarefas, e por outra janela (*GraphBrowser-Window*) onde tarefas, relacionamentos entre tarefas, documentos (nós de contexto e nós terminais), anotações (nós de anotação), e elos de anotação poderão ser definidos e manipulados.

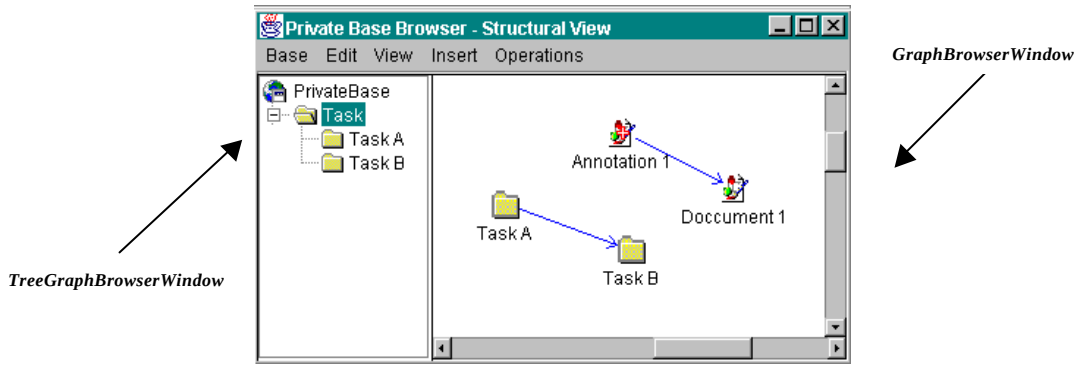


Figura 5.7: Browser de tarefa.

No *browser* de tarefas, elos aparecem como setas, conectando nós (bases privadas, documentos ou nós de anotação) que relacionam; nós de anotação, inicialmente, são apresentados como nós terminais de texto (o que será estendido posteriormente); e documentos serão exibidos como em [Pinto 2000], onde nós de contexto são apresentados

como pastas, e nós de conteúdo, ilustrados com uma gravura específica para cada tipo de conteúdo (texto, imagem, áudio e vídeo).

Tarefas serão representadas por pastas, assim como nós de contexto, que poderão ser abertas ou fechadas para que seu conteúdo[40] possa ser exibido ou não, o que irá evitar, por vezes, uma sobrecarga de informações na visualização da estrutura de um trabalho.

Em uma tarefa poderão ser definidos documentos, nós de anotação, elos de anotação, outras tarefas, para formar a estrutura aninhada de tarefas apresentada, e elos entre tarefas. Os componentes de uma tarefa, na janela *GraphBrowserWindow*, podem ser selecionados pelo usuário, movidos de acordo com sua preferência, e podem ter seus atributos alterados. Quando um componente é selecionado, o nó correspondente aparece em destaque no grafo.

As figuras seguintes ilustram duas visões de um trabalho realizado em uma determinada tarefa (*Task*).

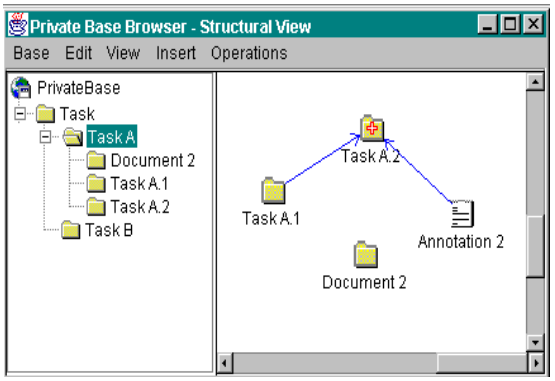


Figura 5.8: Browser de tarefa - exemplo 1.

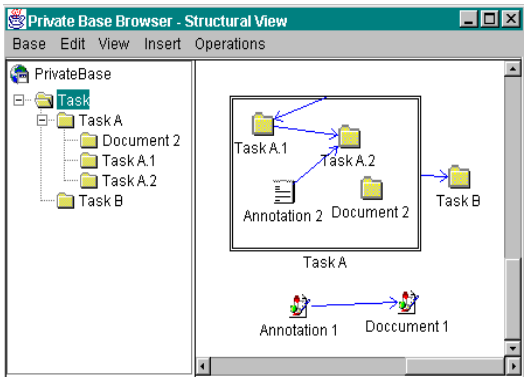


Figura 5.9: Browser de tarefa – exemplo 2.

A Figura 5.8 ilustra o conteúdo de uma tarefa *A* (*Task A*), subtarefa da tarefa *Task*. Na Figura 5-9, os componentes da tarefa *Task* são apresentados de maneira que podem ser vistos, ainda, os componentes de suas subtarefas, quando devidamente expandidas, como é o caso da tarefa *A* (*Task A*). Essa visão pode ser útil quando se deseja visualizar um

[40] Aqui, o conteúdo de uma tarefa inclui nós de anotação, documentos e subtarefas, que podem ser relacionados através de elos.

determinado trabalho dentro de um contexto maior, ou para que atores de uma subtarefa *BP1* possam acessar documentos de uma tarefa na qual *BP1* está recursivamente contida, realizando operações de *check-out's* ou *move's*.

5.4.2 DEFINIÇÃO DOS COMPONENTES DO MODELO DE COOPERAÇÃO NO BROWSER DE TAREFAS

A definição de tarefas, elos entre tarefas, nós de anotação, elos de anotação, e documentos, no *browser de tarefas*, será feita através de janelas criadas para a definição e manipulação de propriedades de cada um desses componentes.

Para cada tarefa *T*, deverão ser definidos valores para os atributos *nome*, *descrição* e o *conjunto de atores*, inicialmente criado contendo a ator criador da tarefa (o *owner*). Definidos esses atributos, poderão ser adicionados a *T*, subtarefas, documentos, anotações e elos relacionando tarefas e anotações, de acordo com as restrições apresentadas em 5.3.

Elos entre tarefas farão parte do atributo *relações* de uma tarefa *T*, quando representam relacionamentos de causalidade ou restrição entre subtarefas de *T* ou a própria tarefa *T*, e deverão ser criados definindo-se basicamente valores para os atributos *nome*, *relação* e *conjunto de binds*. O atributo *relação* de um elo contido em *T* irá descrever um relacionamento de causalidade ou de restrição, definindo papéis que serão desempenhados por subtarefas de *T* ou por *T*, e especificando como o evento de autoria relativo a essas tarefas irá participar dessa relação. O *conjunto de binds* irá conter as tarefas que deverão desempenhar cada papel especificado no atributo *relação*.

Separar a especificação de um relacionamento, definido no atributo *relação*, dos nós, definidos no *conjunto de binds*, que efetivamente vão interagir em uma relação, irá permitir o reuso de uma relação em elos diferentes. Nesse sentido, algumas *relações* (atributo de elo) foram previamente definidas, o que irá facilitar em muito a definição de elos entre tarefas. Essas relações descrevem as seguintes situações:

- O início de tarefas em paralelo quando uma tarefa for iniciada (o que inclui o caso em que apenas uma tarefa deve ser iniciada) – relacionamento de causalidade;
- O início de tarefas em paralelo quando tarefas forem concluídas (o que inclui os casos em que uma tarefa deve ser iniciada quando uma tarefa é concluída, em que uma tarefa deve ser iniciada quando tarefas forem concluídas, e em que tarefas devem ser iniciadas em paralelo quando tarefas forem concluídas) – relacionamento de causalidade;
- O início de tarefas em paralelo apenas quando tarefas forem concluídas (o que inclui os casos em que uma tarefa deve ser iniciada apenas quando outra é concluída, em que uma tarefa deve ser iniciada apenas quando tarefas forem concluídas, ou, em que tarefas devem ser iniciadas em paralelo apenas quando tarefas forem concluídas) – relacionamento de restrição;
- A alteração do estado do evento de autoria de uma tarefa para um estado tal que indique a conclusão de alguma subtarefa, o que irá depender no estado inicial do evento de autoria da tarefa definida no *conjunto de binds* – relacionamento de causalidade;
- A alteração do estado de uma tarefa para um estado que indique a conclusão de todas as suas subtarefas, o que irá depender no estado inicial do evento de autoria da tarefa definida no *conjunto de binds* – relacionamento de causalidade.

Definida a *relação* que deverá participar de um elo, o *conjunto de binds* pode ser facilmente definido, escolhendo-se quais tarefas irão participar de um relacionamento na própria interface para a definição do elo.

Um nó de anotação *A* poderá ser definido em uma tarefa *T* por um ator, pertencente a *T*, que será o *autor* de *A*. Para cada anotação, deverão ser definidos valores para os atributos *nome*, *descrição* e *conteúdo*. A *descrição* de uma anotação poderá ser uma das descrições já previamente definidas (comentários, respostas a comentários, sugestão de alteração em documentos, sugestão de alteração em tarefas, exemplos, perguntas e respostas a perguntas), ou uma nova descrição, de acordo com a preferência do autor. O *conteúdo* de um nó de anotação irá conter a principio anotações textuais, o

que deve ser estendido, de forma que se possa definir como conteúdo nós terminais e de contexto, como especificado no modelo.

Elos de anotação irão representar um relacionamento simples de causalidade, e irão definir um *hiper-elo*[41], cujo ponto terminal de origem é, obrigatoriamente, um nó de anotação, e o ponto terminal de destino, uma base privada, um documento ou um nó de anotação. O atributo *relação* de um elo de anotação foi definido previamente de maneira a representar esse relacionamento.

Documentos serão definidos de acordo com modelo, como foi apresentado em [Pinto 2000] e estarão contidos na base privada onde foram criados. Durante o trabalho, no entanto, esses documentos poderão ser transferidos entre bases privadas, originar versões, serem alterados na base onde estão contidos, ou mesmo serem alterados em outras bases de trabalho conforme as regras e operações definidas em 5.3.2.1. Nesse sentido, para cada tarefa T foi criado o contexto N_{dcr}, para conter os documentos visíveis a T que deverão ser alterados nessa tarefa, e que não estão contidos em T, ou em um nó de contexto contido em T, ou em um nó de contexto contido por recursão em T. Nesse caso, documentos poderão ser inseridos no contexto N_{dcr} quando, então, poderão ser alterados, se a alteração for permitida[42].

Por último, deve-se observar que o Controle de Acesso especificado para o HyperProp, apresentado no Apêndice A, foi implementado mas ainda não foi inserido nesse ambiente.

Aqui, vale acrescentar que o *Controle de Acesso* foi adicionado ao HyperProp para que regras de acesso pudessem ser definidas independente das regras definidas para cada banco de dados particular. Nesse sentido, o controle de acesso, por exemplo,

---

[41] Um hiper-elo é uma especialização da classe elo que possui pelo menos um evento de seleção associado a um dos elementos (nós) de seu conjunto de pontos terminais de origem.

[42] Uma alteração é permitida quando o documento está em um estado temporário, ou mesmo quando está no estado permanente e se deseja realizar alterações em atributos não versionáveis.

permitirá que, durante a autoria cooperativa em bases privadas, possam ser definidos direitos de acesso diferentes a um mesmo *ator* sobre um mesmo *documento*, ou propriedades de *documentos*, em diferentes bases privadas, enquanto o controle de concorrência permitirá o bloqueio e desbloqueio de propriedades por atores e a visualização de travas e pedidos de travas.

6. TRABALHOS RELACIONADOS

O objetivo deste capítulo é apresentar e discutir alguns sistemas, que se propõem a fornecer suporte à cooperação, dando-se ênfase a definição de determinadas características, muito úteis para a elaboração de um modelo de cooperação.

6.1 COAST E CHIPS

A divisão CONCERT, do GMD-IPSI (German National Research Center for Information Technology – Integrated Publication and Information System Institute), visa desenvolver projetos na área de trabalho cooperativo, com o objetivo de definir ambientes e modelos genéricos direcionados a essa área.

Nesse sentido, um framework para a especificação de aplicações de groupaware síncronas, o COAST (***CO****operative* ***A****plication* ***S****ystems* ***T****ecnology*) [SSKH 1996], foi desenvolvido e vem sendo aperfeiçoado.

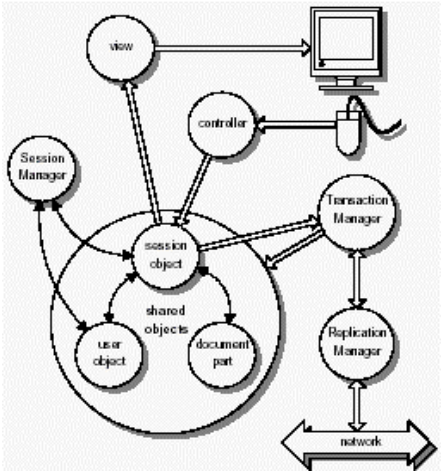


Figura 6.1: Arquitetura do framework COAST.

O COAST define uma arquitetura distribuída e replicada que permite que alterações realizadas em uma aplicação, instância do framework, possam ser propagadas para outras instâncias dependendo do grau de cooperação entre participantes de uma sessão de

trabalho. A última implementação do framework foi feita em VisualWorks SmallTalk (v 3.0).

O framework define basicamente duas camadas: uma camada para o suporte às características básicas de CSCW; e, uma camada específica para a definição do modelo de objetos utilizado durante uma aplicação.

A primeira camada é responsável pela gerência de objetos compartilhados, sendo responsável pela manutenção de réplicas consistentes e pelo acesso a objetos compartilhados por instâncias de uma aplicação. Essa camada fornece controle de transação, um controle de concorrência otimista, baseado em serialização de transações, e atualização automática de visões de um objeto compartilhado, quando esse é alterado. Alterações em objetos compartilhados são notificadas para as diversas instâncias através de um controle de replicação de informações, responsável por replicar ainda informações sobre alterações nas interfaces de usuários.

Ainda nessa camada, o framework define o conceito de sessão de trabalho através de objetos de sessão. Objetos de sessão registram os usuários de uma sessão, definindo o acoplamento e o grau de consciência do trabalho realizado por esses usuários, e os documentos utilizados durante a sessão. Objetos de sessão representam o pilar do modelo de cooperação, através do qual toda a arquitetura foi definida.

Para manipular objetos compartilhados durante uma sessão, foi definida uma interface gráfica com o usuário, o browser de sessão, através da qual participantes poderão manipular objetos, via objeto de sessão.

O modelo conceitual de objetos de sessão se assemelha à idéia de tarefas, no sentido em que ambos representam um determinado trabalho que deve ser realizado por um grupo de atores, agrupando atores, modo de cooperação e documentos utilizados durante o trabalho. Nessa camada, no entanto, ainda não existe a idéia de aninhamento ou ordenação de trabalhos.

Não é especificado, no framework, um mecanismo explícito para o controle de acesso a usuários sobre operações ocorridas durante uma sessão de trabalho, apesar de se identificar essa necessidade.

Como meio de possibilitar o trabalho assíncrono, foi integrado ao COAST, um ambiente de suporte a versões, o VerSE (*Version Suport Environment*) [HaaHi 1996], como interface para um controle de versões.

Especificada para aplicações hipermídia, a segunda camada do framework deve ser instanciada de forma a permitir o trabalho cooperativo sobre um modelo hipermídia (especificado nessa camada). Utilizando os serviços oferecidos pela primeira camada, a camada hipermídia deve definir objetos do modelo hipermídia que será utilizado (basicamente, nós, nós de composição e elos), prover o compartilhamento desses objetos, e definir tipos de objetos de sessão e browsers, equivalentes àqueles definidos na camada anterior.

Baseado em uma instância do framework para aplicações hipermídia, foi definida mais uma camada, em [WanHa 1998], para a definição de trabalhos cooperativos estruturados e organizados. Para dar suporte à definição das novas características acrescentadas por essa camada, foi desenvolvido o CHIPS (*Cooperative Hypermedia Integrated with Process Suport*).

A idéia da nova camada é especificar mecanismos flexíveis para o suporte à coordenação formal, durante a realização de um trabalho, agregando características de sistemas de *workflow* a um sistema de suporte a cooperação hipermídia, utilizando-se os próprios conceitos de nós e elos já definidos no sistema.

Nesse sentido, [WanHa 1998] apresenta um modelo de estruturação de processos muito semelhante ao modelo de cooperação apresentado nesta dissertação.

Processos (*process*) representam um trabalho, que pode ser composto de várias atividades/tarefas (*activities*/*tasks*). Atividades são realizadas por atores (*actors*) aos quais podem ser atribuídos diferentes papéis (*roles*), para especificar responsabilidades. Trabalhos são realizados em áreas de trabalho (*workspaces*), onde estão contidos

documentos utilizados e outras informações públicas ou privadas. Cada atividade é realizada em uma área de trabalho chamada *Activity Space*, que contém informações específicas sobre uma determinada atividade, e processos, em outra área de trabalho chamada de *Process Space*, onde podem ser representados e executados. *Workspaces* representam apenas uma parte de um repositório maior (*Shared Information Space*), onde estão armazenadas informações públicas que poderão ser manipuladas através de operações fornecidas por esse repositório.

Atividades são representadas por nós; o controle e fluxo de dados entre tarefas, por elos; e processos correspondem a nós de composição. As definições de processos e tarefas correspondem à definição de uma estrutura de nós e elos, e a navegação entre processos corresponde a navegação através da estrutura hipermídia.

Um processo tem como elementos básicos tarefas (*task nodes*), elos entre tarefas (*process links*), expressões lógicas associadas a um elo (*transition condition*), que irão determinar se o elo será selecionado ou não, e expressões lógicas associadas a tarefas (*pre- and post conditions*), para determinar se uma tarefa poderá ser iniciada, ou terminada.

Nesse trabalho, são definidos tipos de tarefas, representando: tarefas simples, que não contém outras tarefas; subprocessos, que podem conter outras tarefas; tarefas automáticas, que serão realizadas por um programa; e, tarefas iterativas, para possibilitar loops em processo, já que é definida a restrição de que elos entre tarefas não podem formar grafos acíclicos.

Elos entre tarefas irão carregar referências às informações contidas em espaços de trabalho que devem ser utilizadas por outras tarefas. Nesse sentido, foram definidos diferentes tipos de elos para determinar a relação entre os conteúdos de tarefas relacionadas.

Apesar das definições de tarefas, elos e regras para a execução de tarefas serem muito semelhantes às definições apresentadas nesta dissertação, algumas diferenças podem ser ressaltadas. Nesta dissertação, tarefas foram definidas de forma que subtarefas pudessem ser acrescentadas, ou removidas sem as restrições acrescentadas pela definição

de tipos de tarefas (tarefas simples, subprocessos...), como foi feito em [WanHa 1998]. No que diz respeito à definição de elos entre tarefas, nesta dissertação, elos também definem semânticas de execução, mas a definição sobre o uso de informações compartilhadas entre tarefas, atribuída ao diferentes tipos de elos criados em [WanHa 1998], fica a cargo da definição da própria tarefa, o que torna a especialização de elos hipermídia mais imediata.

Uma funcionalidade interessante apresentada em [WanHa 1998], diz respeito à definição de processos a partir de *templates*. Processos, nesse caso, poderão ser executados a partir de instâncias de *templates*, devidamente preenchidas com as características próprias a cada aplicação, e ainda poderão ter sua estrutura adaptada, para atender as necessidades que vierem a existir durante a sua execução. *Templates* poderão ser criados como estruturas novas, a partir de outros *templates*, ou a partir de instâncias de processo.

A facilidade para definir estruturas reutilizáveis, a partir do modelo de tarefas definido nesta dissertação, é uma característica bastante interessante, que ficará como trabalho futuro.

Durante a execução de um processo, instâncias de tarefas poderão estar no estado ativo (*active*), ou terminado (*finished*), representando a execução de uma tarefa e seu término, respectivamente. Alterações para esses estados geram verificações de pré e pós-condições definidas para cada tarefa, assim como condições de transição definidas em elos.

Durante o trabalho cooperativo, atores poderão se comunicar através de e-mails, documentos contidos em ambientes compartilhados e nós de anotação. Atores, através de interfaces devidamente especificadas, podem ter consciência sobre outros atores trabalhando sobre o mesmo documento, ver informações sobre um documento (seus atributos), e sobre o processo e sobre o estado de tarefas específicas.

6.2 COVER E SEPIA

O CoVer (*Contextual Version Server*) [Haake 1992] é um servidor hipermídia de versões, implementado como uma extensão ao CHS (*Cooperative Hypermedia Server*)

[SHHLS 1992], que, por sua vez, é baseado no HyperBase alemão [SchSt 1990], todos desenvolvidos no GMD – IPSI (German National Research Center for Information Technology – Integrated Publication and Information System Institute).

Aqui vale acrescentar que o CHS e o HyperBase são sistemas definidos para dar suporte à colaboração a nível de armazenamento de dados hipermídia, os quais são usualmente chamados de *hyperbases* (*Hypermidia Databases*). *Hyperbases* têm como objetivo acrescentar determinadas características a sistemas de banco de dados, que usualmente fornecem o suporte ao compartilhamento de informações, integridade, persistência e recuperação de dados. A idéia desses sistemas é separar a camada de armazenamento de dados, fornecida por *hyperbases*, das camadas de aplicação e apresentação, comuns em sistemas hipermídia. *Hyperbases* devem possibilitar a modelagem de estruturas complexas de dados e relacionamentos, o suporte a transações de longa duração, controles de notificação e versionamento, e controle de concorrência, todos esses devidamente adaptados para o suporte ao trabalho cooperativo.

O CHS oferece persistência de nós, elos e composições; fornece meios para a definição de locks em objetos, onde locks são controlados por usuários, compartilhados e persistentes; oferece mecanismos de notificação por objetos, baseado em inscrições persistentes em eventos ocorridos em objetos; e, controle de transação.

O objetivo do CoVer, que forma a infra-estrutura de sistemas como o SEPIA [SHHLSl 1992], é acrescentar características de versionamento ao CHS.

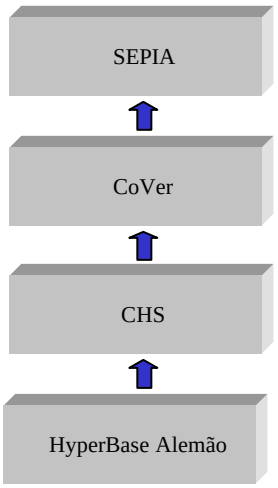


Figura 6.2: Relação entre SEPIA e sistemas antecessores.

O CoVer apresenta um controle de versões, que além de manter o histórico de versões de objetos, relaciona essas informações no contexto de um determinado trabalho. Versões são relacionadas em tarefas (*tasks*), representando esses trabalhos.

Uma tarefa pode ser formada por várias subtarefas, que podem ser interligadas por elos para representar certos tipos de relacionamentos entre tarefas. Elos entre tarefas definem relações sobre a ordem de execução de tarefas, indicando, por exemplo, que tarefas foram ou devem ser realizadas em paralelo ou seqüencialmente.

Em tarefas, o trabalho, no Cover, poderá ser organizado, utilizando os mesmos princípios que foram utilizados para compor o *modelo de tarefas*, apresentado nesta dissertação. Da mesma maneira que o *modelo hipermídia de tarefas*, o Cover trata tarefas como composições (composições no CHS), relacionadas através de elos hipermídia (também utilizando o conceito de elos definido no CHS). Diferentemente do CoVer, no entanto, o *modelo de tarefas* permite a definição de fluxos de trabalhos através da definição da máquina de estados para o *evento de autoria*, e de semânticas definidas em elos, relacionando tarefas.

Tarefas, assim como elos entre tarefas, devem ser definidos de acordo com as necessidades da aplicação rodando sobre o CoVer. Nesse sentido, foram criados dois tipos de tarefas, *tarefas de aplicação* (*application tasks*), que poderão ser definidas de acordo com a necessidade de uma aplicação, e *tarefas de alteração* (*change tasks*), definidas automaticamente pelo CoVer, para criar o histórico de alterações de maneira coordenada. De maneira geral, para cada *tarefa de aplicação*, é definida uma *tarefa de alteração*, onde alterações em versões serão armazenadas pelo CoVer. Para cada *tarefa de aplicação* definida como subtarefa de uma *tarefa de aplicação T*, é criada uma *tarefa de alteração*, identificando o estado corrente de *T* após a realização de sua subtarefa.

Quando uma *tarefa de aplicação* é finalizada em uma certa aplicação, seu estado corrente é congelado pelo CoVer, o que significa o congelamento de versões que representam o trabalho realizado na *tarefa de alteração* corrente. Essas versões, que representam o estado consistente de uma tarefa *T*, são entregues como resultado para uma supertarefa a qual pertence a tarefa *T* (se *T* possui uma supertarefa), enquanto versões

intermediárias são disponíveis apenas para *T* e suas subtarefas. Assim, o CoVer provê uma área para a realização de um trabalho em uma tarefa, escondendo versões inconsistentes de outras tarefas.

Durante a execução de uma tarefa, objetos podem ser incluídos na tarefa, quando automaticamente são congelados, e uma versão é criada, onde alteraões poderão ser feitas.

O Cover permite a definição de anotações, representando comentários e sugestão de trabalho. Anotações foram integradas ao conceito de tarefas, e mantém uma referência à versão anotada (congelada para manter a consistência da anotação) e uma referência à versão considerada resposta ao comentário. Anotações podem ainda conter referência a tarefas geradas a partir de uma anotação sobre uma determinada versão.

Anotações, no CoVer, são utilizadas como meio para catalogar a geração de versões no histórico de versões, e não como veículo de comunicação genérico, como foi proposto no modelo de cooperação apresentado nesta dissertação. Nesse caso, a definição de anotações, no HyperProp, poderia ser estendida, de maneira a possibilitar a definição de anotações no histórico de versões.

O sistema SEPIA (*Structured Elicitation and Processing of Ideas for Authoring*) [SHHLS 1992] é uma ferramenta de suporte a autoria de documentos hipermídia, que define diferentes *espaços de atividades*, cada um com o objetivo de prover um ambiente adequado para cada parte do processo de autoria (planejamento da autoria, definição de conteúdos de documentos, estruturação do documento e preparação do documento final).

Para cada *espaço de atividade*, é definida uma interface apropriada para a manipulação e uso de objetos definidos e necessários a cada uma das atividades. Essa interface é definida através de um *browser de espaço de atividade*, ao qual é associado um objeto de sessão, que armazenam o modo de cooperação entre os usuários participantes de uma atividade, e o conjunto de usuários correntes. Objetos de sessão irão possibilitar a noção de grupo de trabalho.

Alterações a documentos realizadas em um *browsers de espaço de atividade*, de maneira geral, são imediatamente armazenadas em uma base de dados compartilhada, quando notificações são enviadas para cada usuário do grupo de trabalho definido pelo objeto de sessão relativo ao browser. Em um modo de cooperação fortemente acoplado, os *browsers de espaço de atividade* se comunicam diretamente entre si, enviando mensagens sobre alterações em objetos de interface (barras de rolagem, *telepointers*, e alterações no tamanho de uma janela).

Para controlar o trabalho cooperativo sobre um mesmo documento, o SEPIA armazena alterações sobre uma determinada versão *V* realizadas por autores, que não o autor de *V*, em diferentes versões relacionadas. Assim, revisões de objetos são mantidas automaticamente pelo sistema. Ainda, se um usuário desejar salvar um estado particular de um objeto, ele pode ser explicitamente congelado, quando, então, uma nova versão será criada.

Utilizando o conceito de tarefas oferecido pelo CoVer, versões de objetos, relacionadas pala realização de um trabalho, estarão organizadas em tarefas. Para cada projeto definido, uma tarefa será crida automaticamente, com o objetivo de manter o histórico de desenvolvimento do projeto, onde ainda poderão ser definidas outras subtarefas de acordo com as necessidades de um trabalho (*tarefas de aplicação*). O controle de versão nessas tarefas, é mantido pelo CoVer, através de tarefas de alteração, como foi dito anteriormente.

Para possibilitar a criação e manipulações de tarefas em um projeto é definido, para cada projeto, um *browser de tarefas*, onde é destacada a tarefa corrente. Semelhante ao *browser de tarefas* definido nesta dissertação, um usuário poderá selecionar a tarefa que desejar, quando são destacados os objetos utilizados durante a tarefa.

Como o CoVer, o SEPIA não define mecanismos para a definição de fluxos de trabalho entre tarefas, embora seja possível a criação de elos entre tarefas para indicar a seqüência de realização de um determinado trabalho.

6.3 HYPERFORM E HYPERDISCO

O sistema Hyperform (*extensible **Hyper**media Plata**form***) [WiiLe 1997] é um *hyperbase*, desenvolvido na Universidade de Aalborg, na Dinamarca. O HyperForm além de implementar as necessidades básicas de suporte à colaboração a nível de armazenamento de dados hipermídia, ainda fornece características de extensibilidade, permitindo que modelos de dados, assim como novas operações possam ser adicionados em tempo de execução. A idéia é prover suporte a aplicações cooperativas, proporcionando extensibilidade através de especificações definidas em uma linguagem de programação[43].

O sistema Hyperform, ilustrado na Figura 6.3, é constituído por três componentes:

- HBMS (*Hyperbase Management System*);
- Integrador de Ferramentas (*TI*);
- Editores (aplicações de maneira geral).

O componente HBMS fornece uma biblioteca de classes que podem ser especializadas por herança múltipla para formar outras classes com diferentes configurações e funcionalidades. Essas classes do sistema provêem serviços para controle de concorrência, baseado em locks controlados por usuários, compartilhados, de fina granularidade (por atributo), e persistente; controle de notificação de fina granularidade (por atributo), por inscrições persistentes; funcionalidades de busca e pesquisa; controle de versões; e, um controle de acesso. O HBMS deverá incluir, ainda, as definições de âncoras, elos, nós de conteúdo e composição, para uma instância específica, herdando das classes pré-construídas, as características desejadas.

[43] No caso, o Hyperform utiliza a linguagem Scheme [Sprin 1990], um dialeto do Lisp [WinHo 1988]. É bom observar que essa linguagem foi disponibilizada para definições de extensões ao Hyperform, e utilizada para a implementação do próprio sistema, apesar de módulos críticos terem sido implementados em C por questões de desempenho.

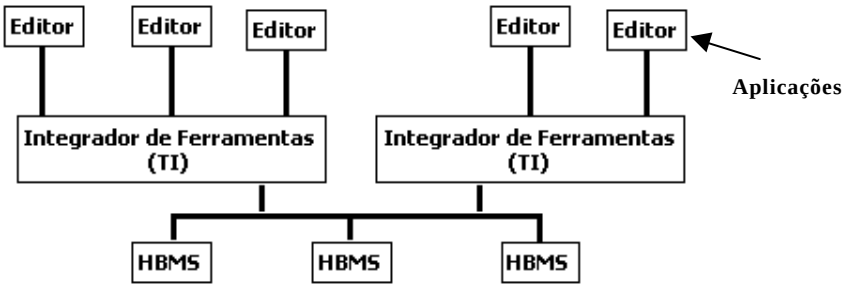


Figura 6.3: Componentes do sistema Hyperform

O TI age como uma interface entre editores e o HBMS, provendo serviços para a criação de relacionamentos (elos) entre objetos e um protocolo para a comunicação com esse serviço e com diferentes HBMS, quando necessário. A grande vantagem do TI é que ele pode manter uma representação compartilhada de estruturas ou objetos armazenados no HBMS (através de eventos), que pode ser utilizada por diferentes editores.

Novos serviços podem ser criados pela especialização e extensão das classes pré-construídas do HBMS e do TI.

O controle de concorrência (*CC object*) é baseado em travas (locks), que podem ser atribuídas a atributos ou a objetos como um todo. Travado, um objeto poderá ser lido, permitindo, assim a navegação por uma rede de objetos, contendo objetos bloqueados. O objeto *CC* ainda provê transações (baseadas no protocolo "2PL - two phase lock"), para que locks possam ser recuperados em casa de falhas no servidor.

O controle de acesso (*AC object*) é influenciado pelos mecanismos de acesso do Unix. O objeto *AC* suporta três diferentes níveis de proteção: *get*, *set* e *delete*, que correspondem às ações básicas para objetos, que serão atribuídas a usuários, grupos de usuários e outros (user/group/others). Uma vez que um modelo de dados for acrescentado ao HBMS, outros direitos podem ser adicionados, como por exemplo direitos para realizar anotações em documentos.

O controle de notificação (*NC object*) é baseado em inscrição em eventos: as inscrições são persistentes e armazenadas em uma lista interna do sistema, sendo especificadas utilizando-se a linguagem Scheme. Cada elemento dessa lista contém informações sobre o usuário inscrito e a expressão em Scheme usada para disparar

eventos. Essas expressões permitem o acesso a todas as variáveis e funções do Hyperform. Esse mecanismo de eventos pode ser utilizado para tornar o servidor HBMS ativo, onde operações específicas são capazes de disparar outras operações, as quais, por exemplo, podem atualizar variáveis e objetos no servidor HBMS automaticamente.

O controle de versões (*VC - objetc*) é bastante simples e é inspirado no sistema RCS [Tichy 1985], onde um determinado objeto tem sua árvore de versões (histórico) armazenada. Versões são manipuladas por meio de operações de *check-out* e *check-in*, podendo ser armazenadas como completas (complete version) ou na forma de deltas (delta version). Nesse último caso, o armazenamento de uma versão *V'* derivada de *V*, implica armazenar as diferenças de *V'* em relação a *V*. O Controle de versões atua no versionamento de um objeto simples, não considerando versionamento, por exemplo, de objetos compostos, como nós de composição.

O sistema HyperDisco (***Hyper****media platform for* ***dis****tributed* ***co****llaborative computing environments*) [WiiLe 1997a], também foi desenvolvido na Universidade de Aalborg, e irá permitir que ferramentas distribuídas e heterogêneas utilizem, de maneira integrada, facilidades hipermídia de um *hyperbase*. O objetivo do projeto é prover uma plataforma hipermídia flexível e extensível para aplicações cooperativas. Para isso, o HyperDisco foi implementado utilizando as funcionalidades já oferecidas pelo HyperForm.

A estrutura do HyperDisco é formada por duas camadas distintas, compatíveis com as camadas definidas no HyperForm: a camada do modelo de dados, que provê os serviços básicos para o armazenamento e manipulação de dados hipermídia, chamados nesse trabalho de *workspaces* (representando o componente HBMS); e, a camada do modelo de integração (representando os integradores de ferramentas - *TI's*), que irá permitir a definição de relacionamentos entre documentos contidos no HBMS através da definição de âncoras e links. Cada *TI* terá associado um *workspace* padrão, para armazenar as classes utilizadas pelo integrador.

A figura seguinte ilustra a estrutura do HyperDisco, onde as aplicações que irão utilizar o sistemas são representadas pelos elementos *Tools*.

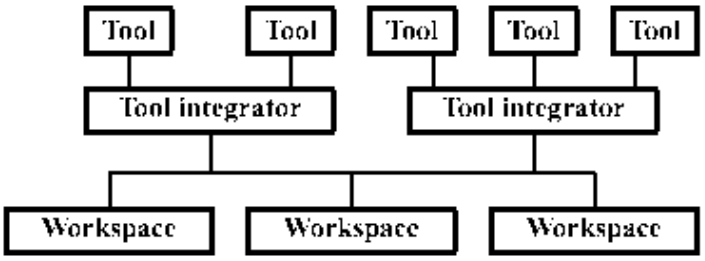


Figura 6.4: Estrutura do HyperDisco.

O HyperDisco provê como componentes de um *workspace*, os objetos âncora, nó, elos, e composições, sobres os quais podem ser aplicados os controles de concorrência, notificação, versão, e acesso.

As funcionalidades básicas para controles de concorrência, notificação, versão, controle de acesso, busca e pesquisa, fornecidas por *workspaces* são aquelas especificadas pelo HyperForm.

No que diz respeito ao controle de concorrência, o HyperDisco utiliza locks controlados por usuários combinados com pequenas transações. Uma sessão de atualização é iniciada quando um pedido explícito de lock é feito, e cada operação nessa sessão (como por exemplo, constantes salvamentos) é realizada como uma pequena transação. A definição de eventos e do controle de notificação permite que ferramentas (tools) possam monitorar alterações realizadas em objetos compartilhados. O controle de concorrência e o controle de notificação definidos operam a nível de atributos, possibilitando que atributos possam ser travados independente de outros, ou que notificações possam ser enviadas sobre operações em determinados atributos.

O controle de versão irá permitir que participantes possam explicitamente fazer pedidos de versionamento em objetos, não existindo nenhum tipo de operação de versionamento automáticas. O controle de acesso é mantido em uma lista de controle de acesso, proporcionando restrições de leitura, escrita, remoção e de anotação a indivíduos ou grupos de indivíduos. O controle de acesso e versão atuam a nível de objetos.

Como meio de restringir um trabalho a um grupo de participantes, os *workspaces* mantém uma lista de usuários que podem acessá-lo. Nesse sentido, *workspaces* podem ser

privados, pertencentes a um grupo, ou públicos. Cada usuário, participante de uma autoria cooperativa, pertencente a um grupo de trabalho, poderia se conectar a um determinado *TI*, definido com uma conexão "*default*" para o *workspace*, e através dele compartilhar objetos, de forma coordenada, pelo controle de acesso, concorrência e versão, e recebendo as devidas notificações. Uma determinada ferramenta (*tool*) poderá mudar o *workspace* utilizado, ou mesmo criar um novo *workspace* para a aplicação. É interessante ainda que usuários podem acessar *workspaces* ou criar elos entre *workspaces* diferentes, se tiver permissão para isso.

Apesar de ser possível as definições de ambientes de trabalho, não foi definido um suporte para a organização do trabalho através da estruturação de ambientes aninhados, e muito menos de regras para a execução de um determinado trabalho.

6.4 WEBDAV

O grupo de trabalho do IETF (Internet Engineering Task Force), denominado WebDAV (Web Distributed Authoring and Versioning), desenvolve extensões ao protocolo HTTP (Hypertext Transfer Protocol) para suportar autoria cooperativa de documentos na World Wide Web [WBDAV 2000].

Hoje em dia, existem várias ferramentas heterogêneas com o mesmo objetivo, no entanto, por não seguirem um padrão, são incompatíveis entre si. A idéia principal do WebDAV é criar um protocolo padrão e funcional para a edição cooperativa de hiperdocumentos na Web.

Até agora, o grupo produziu uma única RFC [WBDAV 1999], que descreve extensões ao HTTP/1.1, que provê um conjunto coerente de métodos, cabeçalhos, formatos de mensagens para operações sobre:

- Propriedades: descrevem recursos de qualquer tipo. Propriedades podem representar autores de um recurso, seu título, restrições sobre o uso de um recurso, etc. Deve ser possível criar, modificar, ler e remover propriedades.

- Coleções: representam recursos que contém outros recursos, diretamente ou por referência. Um mesmo documento pode pertencer a várias coleções, seja diretamente ou por referência. Deve ser possível criar uma coleção, listar e adicionar recursos contidos em uma coleção, e remover recursos de uma coleção.
- Travas (*Locks*): representam o bloqueio de um recurso. Travas podem ser exclusivas, quando apenas um usuário tem permissão para acessar um determinado recurso, ou compartilhadas, quando permitem que mais de um autor alterem um mesmo recurso. Nesse último caso, autores devem estar conscientes sobre possíveis alterações por outros autores. São definidas operações para criar travas, remover travas, verificar se um recurso está travado, e caso esteja, para verificar o autor da trava.
- Operações em Espaço de Nomes (*Namespace Operations*): São definidas as operações para manipular bases de documentos: copiar, mover e remover recursos, criar coleções, e para buscar propriedades e recursos. Nesses espaços, recursos são definidos hierarquicamente como em sistemas de arquivo.

É utilizada a linguagem XML[44] [BraPa 1998] para codificar parâmetros (entrada) e respostas (saída) de métodos, provendo vantagens como extensibilidade e internacionalização na entrada e saída de dados.

Como foi dito anteriormente, o controle de concorrência definido pelo WEBDAV é baseado em um controle de travas de escrita, criadas para evitar que autores realizem atualizações conflitantes e determinados recursos. O bloqueio de um recurso, todavia, não inclui o bloqueio de suas propriedades que poderão ser alteradas quando necessário, excluindo às propriedades controladas automaticamente por um servidor. Uma vez que um recurso é bloqueado por usuários, quando um *token*, identificando o usuário com permissão para alterar o recurso, é gerado, o usuário poderá inclusive se desconectar da

[44] XML (Extensible Markup Language) é uma meta-linguagem para especificação de linguagens para intercâmbio de documentos estruturados.

rede, e se conectar apenas quando desejar salvar o recurso. *Locks* tem granularidade de recursos.

O controle de acesso definido pelo WebDAV é baseado em listas de acessos (ACL's - *access control list*). Para cada recurso contido em um servidor WebDAV, é associada uma lista (ACE's - *access control entries*), contendo privilégios concedidos ou negados a um determinado usuário. Quando um determinado usuário submete uma operação sobre um determinado recurso, o servidor WebDAV confere as ACE's da lista de acesso para determinar se o usuário tem permissão para realizar a operação.

São definidas permissões para ler um recurso, alterar um recurso, ler e alterar a lista de acesso de um recurso, e ainda para o acesso a um único elemento da lista de acesso: a entrada correspondente ao usuário corrente. Essas permissões poderão, ainda, ser combinadas de acordo com determinadas restrições. É interessante notar que listas de acessos são definidas como propriedades de um recurso, assim como outras novas propriedades definidas para o controle de acesso.

Em relação ao controle de versões, existe um "*draft*" criado pelo grupo que está em um estado avançado, próximo a se tornar uma versão estável. O trabalho de versões foi iniciado pelo grupo WebDAV e atualmente está sendo continuado por um subgrupo chamado Delta-V [Delta-V 2000]. As facilidades principais do controle de versões no DAV incluem o registro do histórico de derivação de objetos, operações de *check-out* e *check-in*, atribuição de características de mutabilidade ou imutabilidade[45] à recursos, fusão de versões, recuperação de uma versão padrão (*default*) de um recurso e versionamento de coleções.

Um controle de notificação não foi ainda definido.

Deve-se notar que operações de locks, assim como outras operações definidas pelo protocolo WebDAV, devem ser implementadas por clientes e servidores que desejam

[45] Os atributos de um recurso imutável não podem ser modificados.

utilizar esse protocolo. O WebDAV não restringe, no entanto, a comunicação entre clientes HTTP simples e servidores WebDAV, possibilitando que diversos indivíduos possam acessar dados em um servidor WWW qualquer, independente se esse implementa o protocolo WebDAV ou não. Nesse caso, no entanto, os mecanismos de coordenação inseridos por esse protocolo não serão válidos, o que poderá originar diversos problemas no que diz respeito à alteração de recursos compartilhados, principalmente durante a autoria cooperativa de documentos hipermídia. No caso de *locks*, uma alteração em um recurso travado não será possível por um cliente simples, mas o cliente, não tendo consciência de travas em recursos, poderá realizar um GET do documento, enquanto o mesmo está travado (o que é sempre permitido), modificá-lo, e atualizar o documento depois que já estiver desbloqueado, sem a consciência de que está sobrescrevendo um documento modificado.

Não existe no WebDAV a noção de grupos de trabalho, ou tarefas, apenas foram definidos mecanismos de suporte a cooperação no que diz respeito à gerência de objetos compartilhados, como em *hyperbases*, onde hiperdocumentos são armazenados, como em sistemas de arquivo, e geridos pelos controles de acesso, concorrência e versões definidos. Nesse sentido, o WebDAV ainda especifica o protocolo de comunicação entre clientes e servidores WebDAV.

6.5 RESUMO

O suporte ao trabalho cooperativo, apresentado nesta dissertação, baseou-se em uma estrutura hierárquica de tarefas observada em [WanHa 1998] e [Haake 1992], para organizar o trabalho que deverá ser realizado em cooperação. A partir dessa estrutura, foi criado o *modelo de tarefas* quando foram incluídos relacionamentos entre tarefas, como em [WanHa 1998], para representar o processo de execução de trabalhos seguindo a idéia de *sistemas de workflow* [Fisch 2001] e *StateCharts* [Harel 1997].

Aos trabalhos apresentados em [WanHa 1998] e [Haake 1992], foram acrescentadas a idéia de *evento de autoria*, para representar eventos ocorridos em tarefas durante a sua execução; e estados de *evento de autoria* para representar os estados de

tarefas durante a execução de um trabalho. Essas características possibilitaram alterações sobre o processo de execução de tarefas, de acordo com certas observações feitas na Seção 2.2.5, de maneira a possibilitar adições e remoções de subtarefas e relacionamentos em tarefas. Nesse sentido, o modelo apresentado em [WanHa 1998] limita a alteração de tarefas, já que tarefas são tipadas, e definidas de acordo com características particulares.

A partir do *modelo de tarefas* foram definidos os mecanismos: de colaboração, baseado em ambientes particulares ou públicos (que restringem a visibilidade de documentos e alterações em documentos), seguindo os mesmos princípios apresentados em [WanHa 1998], [Haake 1992] e [WiiLe 1997]; de comunicação, através de anotações, utilizadas também em [WanHa 1998] e [Haake 1992], todavia acrescentando algumas funcionalidades (anotações sobre anotações, e anotações sobre tarefas), e controles de notificação (definidos nos *hyperbases*); de coordenação, através dos controles de acesso e concorrência, definidos como nos *hyperbases*, e da definição de papéis a atores de tarefas, como em [WanHa 1998]. Baseado nesses mecanismos, foi definido *o modelo de cooperação* como suporte à autoria cooperativa de documentos.

O *modelo de cooperação* reúne algumas das características apresentadas pelos hyperbases, ao mesmo tempo em que inclui as características de divisão e organização de um trabalho em tarefas relacionadas, muito semelhante a idéia do modelo utilizado no CHIPS.

No que diz respeito à colaboração em tarefas, uma característica difere o *modelo de cooperação*, do apresentado em [Haake 1992]. O trabalho em [Haake 1992] é sempre baseado em versões de dados, o que irá possibilitar o armazenamento eficiente do histórico de um trabalho, mas que muitas vezes poderá gerar versões inúteis, ou mesmo indesejadas. Não é possível no CoVer, e conseqüentemente no SEPIA, por exemplo, que um determinado atributo possa ser alterado em uma única versão por autores definidos. Nesse sentido, o NCM permite a definição de atributos não versionáveis, que podem ser alterados sem que, no entanto, uma nova versão seja criada, como desejado.

Por último, uma característica interessante definida em [WanHa 1998], como foi dito, diz respeito ao reuso de estruturas para a execução de um determinado trabalho, como templates, o que fica, aqui, como trabalho futuro.

7. CONSIDERAÇÕES FINAIS

Este trabalho teve como propósito a definição de um ambiente de suporte à autoria cooperativa, levando-se em conta necessidades relevantes à colaboração, comunicação e coordenação do trabalho realizado por um grupo de pessoas.

Como suporte a esse ambiente, foi elaborado um *modelo de tarefas*, onde tarefas representam trabalhos que podem ser organizados e executados por um grupo de atores segundo uma estrutura de tarefas. Baseado nesse modelo, certos conceitos fundamentais, que dizem respeito à colaboração, comunicação e coordenação, foram apresentados e definidos em um *modelo de cooperação*, para compor as características fundamentais e necessárias a um sistema que se disponha a oferecer suporte à autoria cooperativa de documentos.

Para possibilitar a definição de regras de acesso durante a autoria cooperativa, como forma de coordenação do trabalho, ainda foi especificado um *Controle de Acesso* que, definido de forma genérica e adaptável, pode ser especializado para diferentes sistemas que necessitem acrescentar essa características rapidamente.

Por último, o *modelo de cooperação* foi especializado no *modelo de cooperação* específico para o HyperProp, quando foram utilizados determinados objetos do modelo conceitual NCM na sua composição. Para possibilitar a definição dos conceitos introduzidos por esse modelo, ainda foi implementado um ambiente de suporte à autoria cooperativa no HyperProp.

7.1 CONTRIBUIÇÕES

Este trabalho foi realizado identificando-se as necessidades inerentes à cooperação entre indivíduos, ao mesmo tempo em que determinadas características oferecidas por sistemas hipermídia foram observadas. O resultado final foi a especificação de um modelo para o suporte à cooperação (*o modelo de cooperação*) e a sua especialização em um modelo de

suporte à cooperação baseado em certos conceitos usuais em sistemas hipermídia, mais especificamente utilizando-se conceitos hipermídia do NCM.

As definições e discussões a respeito do *modelo de tarefas* e do *modelo de cooperação* devem esclarecer as necessidades inerentes a um sistema que suporte a cooperação entre atores, proporcionando meios para a colaboração, comunicação e coordenação do trabalho. A especialização desse modelo utilizando-se conceitos hipermídia ainda tornou possível, e fácil, a inserção dessas necessidades em sistemas hipermídia, o que pôde ser visto para o sistema HyperProp.

Vale ressaltar aqui, que foram criados meios para a alteração de uma estrutura de tarefas (considerando-se determinadas observações e restrições), o que possibilita a adaptação de um processo de execução de um trabalho. Esse fato é importante para aqueles trabalhos que não podem ser estruturados de antemão por completo, como é o caso de um processo de autoria cooperativa de documentos.

De maneira geral, em sistemas de *workflow*, processos são planejados, definidos (através de determinadas linguagens de definição de processo), e depois de definidos e testados deverão ser instanciados, quando sua execução poderá ser iniciada. Todavia, depois de instanciados, processos de *workflow*, de maneira geral, não podem ser alterados o que, como visto anteriormente, pode ser muito útil em alguns casos. Nesse caso, para que novas características sejam inseridas em um processo, a sua definição deverá ser adaptada para que uma nova instância possa ser criada com as características desejadas.

O *modelo de tarefas* poderia ser visto como uma alternativa a modelos de *workflow*, para aqueles sistemas que desejem acrescentar mecanismos flexíveis para a definição de processos de execução de trabalhos, e alteração dos mesmos. Para um sistema hipermídia, essa alternativa torna-se ainda mais interessante, já que o *modelo de tarefas* poderá ser especificado utilizando-se conceitos do próprio sistema.

Para que o *modelo de tarefas* e o *modelo de cooperação* fosse especificado utilizando-se conceitos do modelo NCM foram feitas algumas adaptações a objetos já existentes, assim como foram acrescentadas ao modelo novas características, como pôde

ser visto nas Seções 5.3.1 e 5.3.2. Foram definidos: os conceitos de organização de bases de trabalho, segundo uma estrutura aninhada; relacionamentos entre essas bases, para que um trabalho pudesse ser organizado, estruturado e executado segundo uma determinada ordem preestabelecida; o *evento de autoria*; o conceito de visibilidade de documentos e tarefas; além de primitivas para a troca de informação entre bases de trabalho.

A definição do ambiente de cooperação para o HyperProp foi outra contribuição fundamental para o suporte a autoria cooperativa de documentos NCM. Esse ambiente possibilitou a definição e manipulação dos conceitos acrescentados ao HyperProp, pelo *modelo de cooperação*, proporcionando meios para a visualização e execução adequada de um trabalho realizado segundo uma estrutura de tarefas.

Como última contribuição, no que diz respeito a coordenação de um trabalho, foi definido um *Controle de Acesso* genérico, o qual foi especializado para o HyperProp, de forma a possibilitar a definição de regras de acesso a propriedades e entidades NCM.

7.2 TRABALHOS FUTUROS

O modelo de tarefas permitiu que um determinado trabalho pudesse ser estruturado e organizado através da definição de uma estrutura de tarefas. Nesse sentido, um trabalho futuro seria a adição de novas características a esse modelo para que uma determinada estrutura de tarefas pudesse ser reutilizada, ou mesmo especializada, na definição de uma outra estrutura. Uma das alternativas para que essa característica possa ser alcançada leva a um outro trabalho futuro: estudar como o conceito de *"estilos arquiteturais"*[46], oferecido em ADLs (*Architecture Definition Language*) [ShaGa 1996], poderia ser utilizado na definição de *templates* para tarefas.

[46] *Estilos arquiteturais* visam fornecer um conjunto de características e restrições que toda configuração que segue um determinado estilo deve satisfazer.

Para acrescentar essa nova característica a sistemas hipermídia, deve-se ainda estudar como o conceito de *estilos arquiteturais* poderia ser generalizado para o domínio hipermídia.

No que diz respeito às definições dos mecanismos que irão possibilitar a colaboração, comunicação e coordenação durante a cooperação no HyperProp, ficam como trabalhos futuros: a definição de novas regras de notificação e, possivelmente, a adaptação de regras já definidas em [Muniz 2000], para possibilitar notificações relevantes durante a realização de um trabalho cooperativo em bases privadas; e a definição de direitos de acesso a partir de papéis definidos para atores.

A implementação do ambiente de cooperação para o HyperProp foi um passo inicial, que possibilitou a criação e manipulação dos conceitos fundamentais e necessários à cooperação, que deverá ser estendido para que determinadas características ainda sejam acrescidas ao ambiente.

Entre essas características estão: a comunicação com o controle de notificação, para que, no ambiente de trabalho, notificações possam ser visíveis a atores, e regras de notificação possam ser definidas e manipuladas; e a definição de regras que restrinjam a visibilidade de tarefas no ambiente de trabalho, para que um usuário, ator em uma determinada tarefa, só tenha consciência de documentos e tarefas visíveis a ele. Ainda, características do Controle de Acesso deverão ser inseridas ao ambiente de trabalho para que regras de acesso possam ser definidas e alteradas durante a execução do trabalho. Deverá ser possível, também, a visualização de travas e pedidos de trava nesse ambiente, o que será fundamental para a consciência de atores engajados em um mesmo trabalho cooperativo.

A identificação, definição e inserção de novas necessidades a esse ambiente, ficam como trabalhos futuros.

Esse ambiente, todavia, é apenas um passo para a implementação de um sistema que deverá possibilitar a autoria cooperativa de documentos hipermídia, no HyperProp, por atores possivelmente distribuídos em localidades diferentes. Nesse sentido, muito ainda tem

para ser desenvolvido, a começar pela definição de uma arquitetura para a comunicação entre os diferentes atores participantes de um processo cooperativo.

REFERÊNCIAS BIBLIOGRÁFICAS

[Batis 1994] Batista, *T*. V. "Controle de Versões no Modelo Hipermídia de Contextos Aninhados". Dissertação de Mestrado do Departamento de Informática da PUC-Rio, Agosto 1993.

[BoRuJ 1999] Booch, G.; Rumbauch, J.; Jacobsen, I. "UML - The Unified Modeling Language" - Addison Wesley.

[BraPa 1998] Bray, *T*.; Paoli, C. M.; Sperberg-McQueen. "Extensible Markup Language (XML)". World Wide Web Consortium Recommendation REC-xml-19980210.

[CamGo 1998] Campbell, B.; Goodman, J.; "HAM: A general purpose hypertext abstract machine". ACM Computing Surveys, (September), 269-317. 1988.

[CTLRRS 1991] Casanova, M. A.; Tucherman, L.; Lima, M. J.; Rangel Netto, J. L.; Rodriguez, N. R.; Soares, L. F. G. "The Nested Context Model for Hyperdocuments". Proceedings of Hypertext 91, Texas, 1991.

[Delta-V 2000] "IETF Delta-V Working Group: Extending the Web with versioning and configuration management", 2000.

http://www.webdav.org/deltav

[FaySJJ 1999] Fayad, M.; Schimidt, D. C.; Johnson, R. E. "Building Aplication Frameworks – Object-Oriented Foundations of Frameworks Design". Editora: Wiley, 1999.

[Ferbe 1999] Ferber, J. "Multi-agent Systems: an Introduction to Distributed Artificial Intelligence". Editora: Addison-Wesley, 1999.

[Ferre 1995] Ferreira, A. B. H. "Aurélio: Novo Dicionário Básico da Língua Portuguesa". Editora Nova Fronteira SA, 1995.

[Fisch 2001] Fischer, L. "WorkFlow book 2001", published in association with the Workflow Management Coalition (WfMC), October 2000.

[FuksM 1994] Fuks, H.; Moura, L. M. "A Document Based Approach for Cooperation". Journal of Brazilian Computer Society, Number 1, Vol 1, July 1994.

[GaHJV 1994] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. "Design Patterns - Elements of Reuseble Patterns", Addison Wesley, 1994.

[GreRo 1999] Greenberg, S.; Roseman, M. "Groupware Toolkits for Synchronous Work". Computer-Supported Cooperative Work, Trends in Software Series 7, John

Wiley & Sons, 1999.

[GrHMS 1994] Gronbaek, K.; Hem, J. A.; Madsen, O. L.; Sloth, L. “Cooperative Hypermedia Systems: a Dexter-based architecture. Communications of the ACM, 37(2), Fevereiro, 1994.

[Gronb 1997] Gronbaek, K. “Designing Dexter-based Hypermedia Services for the World Wide Web”. Proceedings of the ACM Conference on Hypertext, 1997.

[HaaHa 1993] Haake, A. ; Haake J. M. “Take CoVer: Exploiting Version Support in Cooperative Systems.” Proceedings of the InterCHI '93, Amsterdam, Netherlands, April 26-29, 1993.

[HaaHa 1994] Haake, A.; Haake J. M. “Under CoVer: The implementation of a Contextual Version Server for HyperText Applications.” Proceedings of ACM, 1994.

[HaaHi 1996] Haake, A., Hicks, D. “VerSE: Towards hypertext versioning styles.” Proceedings of the 7th ACM Conference on Hypertext (HT ‘96) , Washington, D.C., USA, March 16-20, 1996.

[Haake 1992] Haake, A. “CoVer: A Contextual Version Server for Hypertext Applications”. Proceedings of the 4th ACM conference on Hypertext, 1992.

[HaaWa 1997] Haake, J. M.; Wang, W. “Flexible Support for Business Processes: Extending Cooperative Hypermedia with Process Support. Proceedings of the International Conference on Supporting on Supporting Group Work: the Integration Challenge, 1997.

[HalSc 1994] Halasz, F.; Schwartz, M. “The Dexter Hypertext Reference Model (Edited)”. Communications of the ACM, 37(2), Fevereiro, 1994.

[Harel 1987] Harel, D.; "Statecharts: a Visual Formalism for Complex Systems," Science Computer Program, Vol 8, pp 231-274, 1987

[HaWil 1992] Haake, J.; Wilson, B. “Supporting Collaborative Writing of Hyperdocuments in SEPIA”. Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW ‘92), 1992.

[JAVA 2000] “The Java Programming Language”, 2000.

http://java.sun.com

[Lima 1997] Lima, G. M. de A.; Toledo M. B. F. de; “Um modelo de transações Cooperativas Integrado a um modelo de Versões”. XII SBBD, 1997.

[LucMa 2000] Lucena, C.J.P; Markiewicz M.E. “Understanding Object Oriented Framework Engineering”. PUC-Rio Inf.MCC39/00 October, 2000.

[MHJPR 1995] Mathur, A. G.; Hall, R. W.; Jahanian, F.; Prakash, A.; Rasmussen, C. "The Publish/Subscribe Paradigm for Scalable Group Collaboration Systems". Department of Electrical Engineering and Computer Science, University of Michigan, Technical Report, 1995.

[MinMa 1993] Minor, S.; Magnusson, B. "A Model for Semi-(a)Synchronous Collaborative Editing". Proceedings of the Third European Conference on Computer Supported Cooperative Work, Kluwer Academic Publishers, 1993.

[MucSo 2001] D.C. Muchaluat-Saade; L.F.G. Soares. "Hypermedia Spatio-Temporal Synchronization Relations also Deserve First-Class Status", Technical Report of the TeleMidia Laboratory, Computer Science Department, PUC-Rio, May 2001, also submitted to MMM'2001, Amsterdam, November 2001.

[MuLZS 2000] Muntz, R. R.; Li, D.; Zhou, L.; Sun. C. "Operation Propagation in Real-Time Group Editors". IEEE MultiMidia October-December 2000

[Muniz 2000] Muniz, Bruno Cavalcanti; "Notificação e Controle de Versões para o Suporte à Autoria Cooperativa no Sistema HyperProp". Dissertação de Mestrado do Departamento de Informática da Puc-Rio, Julho, 2000.

[OHS 1999] "Open Hypermedia Systems Working Group".

http://www.ohswg.org, 1999.

[PerLi 1996] Perez Luquez, M.J., Little T.D.C. "A Temporal Reference Framework for Multimidia Syncronization". IEEE Journal on Selected Areas in Communications (Special Issue: Syncronization Issues in Multimedia Communication) Vol.14, No. 1,January 1996, pp.36-51.

[Pinto 2000] Pinto, L.A.F. "Autoria Gráfica de Estruturas de Documentos Hipermídia no Sistema HyperProp". Dissertação de Mestrado, Agosto 2000

[POET 2000] POET Java Edition. "POET 6.1 Programmer's Guide", 2000.

[Rodri 1997] Rodrigues, R.F. "Formatação Temporal e Espacial no Sistema HyperProp" *Tese de Mestrado, Departamento de Informática, PUC - Rio*, Brasil, Maio, 1997.

[RoMuS 1998] Rodrigues, R. F.; Muchaluat-Saade, D. C; Soares, L. F. G. "Composite Nodes, Contextual Links and Graphical Structures Views on the WWW". Special Issue on World Wide Web of the Journal of Brazilian Computer Society, Vol. 5, N 2, Nov 1998

[RosGr 1992] Roseman, M.; Greenberg, S. "GROUPKIT: A Groupware Toolkit for Building Real-Time Conferencing Applications". Proceedings of the Conference on Computer Supported Cooperative Work, Outubro-Novembro, 1992.

[SchHa 1993] Schutt, H. A.; Haake, J. M. “Server suport for hypermidia systems”. In Hypermidia Proceedings der Internationalen Hypermidia’ 93 Konferenz, (march - 1993), pg.45-56.

[SchKi 1996] Schuckmann, C.; Kirchner, L. et al. “Designing Object-Oriented Synchronous Groupware with COAST”. Proceedings of the Conference on Computer Supported Cooperative Work, Novembro, 1996.

[SchSt 1990] Schutt, H.A.; Streitz, N. A. “HyperBase: A Hypermedia Engine Based on a Relational Database Management System”. Proceedings of the European Conference on Hypertext, 1990.

[ShaGa 1996] Shaw, M.; Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Abril de 1996.

[SHHLS 1992] Streitz, N. A.; Haake, J. M.; Hannemann, J.; Lemke, A.; Schuler, W.; Schütt, H. A.; Thüring, M. “SEPIA: A Cooperative Hypermedia Authoring Environment”. Proceedings of the 4th ACM Conference on Hypertext (ECHT'92), 1992.

[Soare 2000] Soares, L. F. G. “Modelo de Contextos Aninhandos”, Relatório Técnico do Laboratório Telemídia, Departamento de Informática da PUC-Rio, 2000.

[SoRoM 2000] Soares L.F.G.; Rodrigues R.F.; Muchaluat D.C. "Authoring and Formatting Hypermedia Documents in the HyperProp System". ACM Multimedia Systems Journal, Springer-Verlag, Vol. 8, No. 2, março de 2000, pp.118-134.

[SoSRM 1999] Soares, L.F.G.; Souza, G. L.; Rodrigues, R.; Muchaluat, D. “Versioning Support in the HyperProp System”. Multimedia Tool & Application, Vol. 8, No. 8, 1999.

[Sprin 1990] Springer, G. "Scheme and the Art of Programming". MIT Press and McGraw Hill, 1990.

[SSKH 1996] Schuckmann, C.; Schümmer, J.; Kirchner, L.; Haake, J. “Designing object-oriented syncronous groupware with COAST”. Proceedings of ACM CSCW’96, Novembro, 1996 – pg 30-38.

[StGHH 1994] Streitz, N. A.; Geibler, J.; Haake, J. M.; Hol, J. “DOLPHIN: Integrated Meeting Support across Local and Remote Desktop Environments and Liveboards. Proceedings of the Conference on Computer Supported Cooperative Work, Outubro, 1994.

[TakHi 1996] Takahashi, K. Higuchi, M. “Hypermedia Support for Workflow Management in Collaborative Document Production”. NTT Software Laboratories, 1996.

[TaKYa 2000] Takahashi, K.; Yana, E. “A Hypermedia Environament for Global Collaboration”. IEEE Multimidia October-December 2000.

[Tichy 1985] Tichy, W. “RCS – A System for Version Control”. Software Practice and Experience, Vol. 15, Junho, 1985.

[VODAK 2000] “VODAK, an Open Object-Oriented Database System”. Developed at GMD-IPSI, Datmstadt, Germany. www.darmstad.gmd.de/oasys/projects/vodak/extended.htm

[WanHa 1998] Wang, W.; Haake, J. M. “Flexible Coordination with Cooperative Hypermedia. Ninth ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space Structure in Hypermedia Systems, Junho, 1998.

[WBDAV 1999] RFC 2518. "HTTP Extensions for Distributed Authoring -- WebDAV", 1999.

[WBDAV 2000] “The Web Distributed Authoring and Versioning Working Group”, 2000. http://www.webdav.org

[Weiss 2000] Weiss, G "Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence". Editora: Massachusetts Institute of Technology, 2000.

[WhiGo 1999] Whitehead, E. J. Jr.; Goland, Y. Y. “WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web”, 1999.

[WiiLe 1992a] Wiil, U. K; Legget, J. J. “Hyperform: An Extensible Hyperbase Managemente System”. Technical Report TAMU-HRL 92-003, Texas A&M University, 1992.

[WiiLe 1993] Wiil, U.K.; Legget, J. J. “Concurrency Control in Collaborative Hypertext Systems”. Proceedings of the ACM Conference on Hypertext, 1993.

[WiiLe 1997] Wiil, U. K; Legget, J. J. “Hyperform: A Hypermedia System Development Environment”. ACM Transactions on Information Systems, 15, 1, janeiro de 1997.

[WiiLe 1997a] Wiil, U.K.; Legget, J. J. “Workspaces: the HyperDisco Approach to Internet Distribution”. Proceedings of the Eighth ACM Conference on Hypertext, 1997.

[Will 1993] Wiil, U. K. “Experiences with HyperBase: A Hypertext Database Supporting Collaborative Work”. SIGMOD, 22(4), Dezembro, 1993.

[WilLe 1992] Wiil, U. K.; Legget, J. J. “HyperForm: Using Extensibility to Develop Dynamic, Open and Distributed Hypertext Systems”. ACM Conference on Hypertext, 1992.

[WinHo 1988] Winston, P. H.; Horn, B. K. P. "Lisp". Addison-Wesley Pub Co, 1988.

APÊNDICE A: CONTROLE DE ACESSO NO HYPERPROP

O Controle de Acesso que irá atuar no HyperProp é uma especialização do Controle de Acesso definido no Capítulo 4, para objetos NCM, e será apresentado ao longo desta seção.

Serão apresentados como componentes desse controle o controle de acesso, na Seção 1, e o controle de concorrência, na Seção 2, devidamente especificados para *objetos NCM.*

1 Controle de Acesso

O controle de acesso, definido para o HyperProp nesta seção, irá possibilitar a definição de direitos de acesso a usuários do sistema sobre *entidades NCM*, assim como a *propriedades NCM.*

Esse controle, definido para *entidades* de maneira geral, poderá, no entanto, ser especializado, quando necessário, para cada entidade do modelo NCM, quando poderão ser definidas permissões de acesso próprias a cada uma delas, conforme poderá ser visto na Seção 1.2.

1.1 Módulo Controle de Acesso

Para formar o controle de acesso genérico para o HyperProp, os elementos básicos desse controle – *sujeito, objeto* e *permissão* – foram especializados da seguinte forma:

- O *sujeito*, representado pela classe *Subject*, definida no Framework, será especializado em um objeto *Usuário* (*User*);
- *Objetos* serão especializados em *entidades* ou *propriedades NCM*, como instâncias da classe *GenericObject*; e,

- *Permissões* para *entidades* ou *propriedades*, serão definidas como instâncias da classe *ObjectPermission*.

O módulo controle de acesso será implementado segundo uma especialização do seguinte diagrama de classes, que descreve as classes e relacionamentos do módulo do controle de acesso genérico.

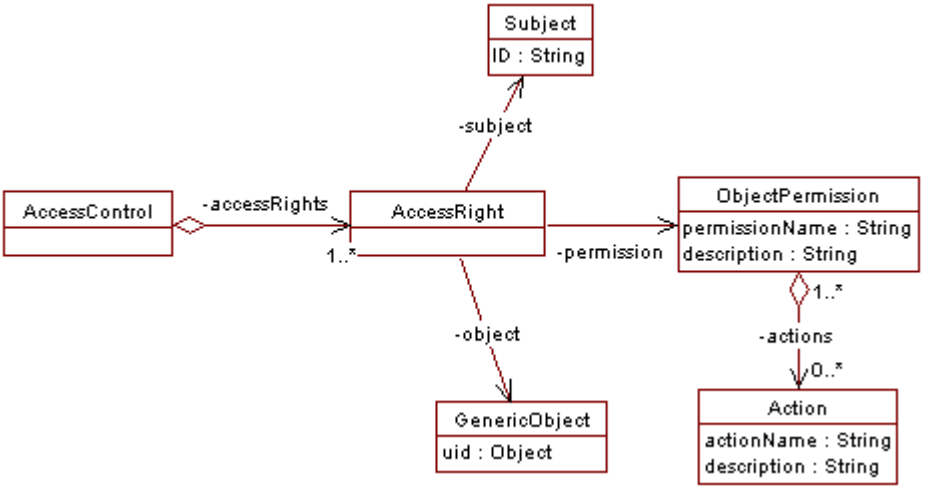


Figura 7.1: Diagrama de classes para o módulo controle de acesso.

A classe *AccessControl* é responsável pela gerência do controle de acesso e possui uma coleção de *direitos de acesso* (*accessRights*). Essa classe fornece os métodos para buscar uma *permissão* de um determinado *sujeito* sobre um *objeto*, alterar *permissões* de *sujeitos* sobre *objetos* (criar novas *permissões* se ainda não existirem, ou remover *permissões*) e para testar se um *sujeito* possui *permissão* para realizar determinadas ações em um objeto. Ainda, foram disponibilizados os métodos para retornar os *objetos* sobre os quais um *sujeito* possui *permissão* e para retornar os *sujeitos* que têm *permissão* sobre um determinado *objeto*.

A classe *AccessRight* representa o direito de acesso propriamente dito, relacionando uma *permissão*, um *objeto* e um *sujeito*.

As classes *GenericObject*, *Subject* e *ObjectPermission* representam as classes básicas do Controle de Acesso. As duas primeiras classes são pontos de flexibilização, e deverão ser instanciadas para cada *objeto* e *sujeito*, respectivamente, que desejar participar do controle de acesso.

A classes *AccessControl* deve ser especializadas para cada tipo de objeto que deve ser controlado, e assim o será, para entidades e propriedades NCM.

MÓDULO CONTROLE DE ACESSO ESPECÍFICO PARA O NCM

Para acrescentar agilidades na consulta de uma permissão atribuída a um *usuário* sobre um *objeto*, a estrutura de armazenamento de direitos de acesso (*AccessRight*), definida em *AccessControl* como uma coleção simples de direitos de acesso, foi especializada em uma lógica de *hashtables*.

Para cada *objeto*, controlado pelo *controle de acesso*, serão armazenados os *sujeitos* com permissão sobre esse *objeto*, e para cada um desses *sujeitos*, será armazenado o *direito de acesso* propriamente dito.

A lógica de armazenamento de direitos de acesso para um objeto qualquer, seguindo uma estrutura de *hashtables*, pode ser vista na figura seguinte:

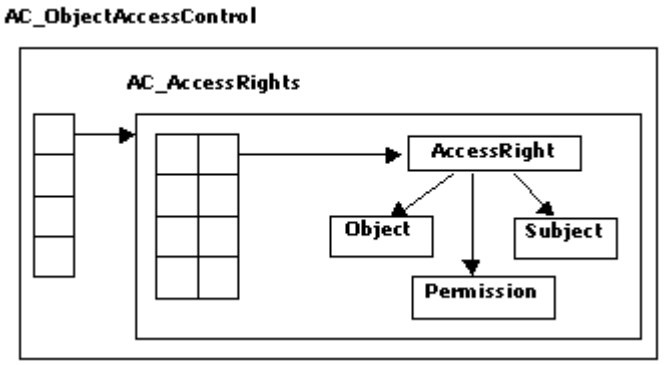


Figura 7.2: Estrutura de armazenamento de direitos de acesso.

Um objeto *AC_ObjectAccessControl*, uma especialização do objeto *AccessControl*, possui uma tabela para armazenar direitos de acesso por *objetos*. Para cada entrada (*objeto*) dessa tabela, serão armazenados os direitos de acesso de todos os sujeitos que possuem alguma permissão sobre o *objeto*, em um objeto *AC_AccessRights.*

O objeto *AC_AccessRights* possui, por sua vez, uma tabela para armazenar direitos de acesso por sujeitos. Para cada entrada (sujeito) dessa tabela, será armazenado o objeto *AccessRight*.

Essa especialização da estrutura de armazenamento de direitos de acesso facilitará consultas sobre quais sujeitos tem permissão sobre um determinado objeto, dificultando, no entanto, consultas sobre quais objetos um determinado sujeito tem permissão, todavia não mais que a estrutura genérica de armazenamento definida anteriormente. Julgou-se que essa especialização para o HyperProp fosse suficiente, já que normalmente consultas serão feitas sobre objetos, e não sobre sujeitos.

Definida a estrutura de armazenamento de direitos de acesso, os pontos de flexibilização do módulo Controle de Acesso foram instanciados para entidades e propriedades NCM, como na Figura 7.3.

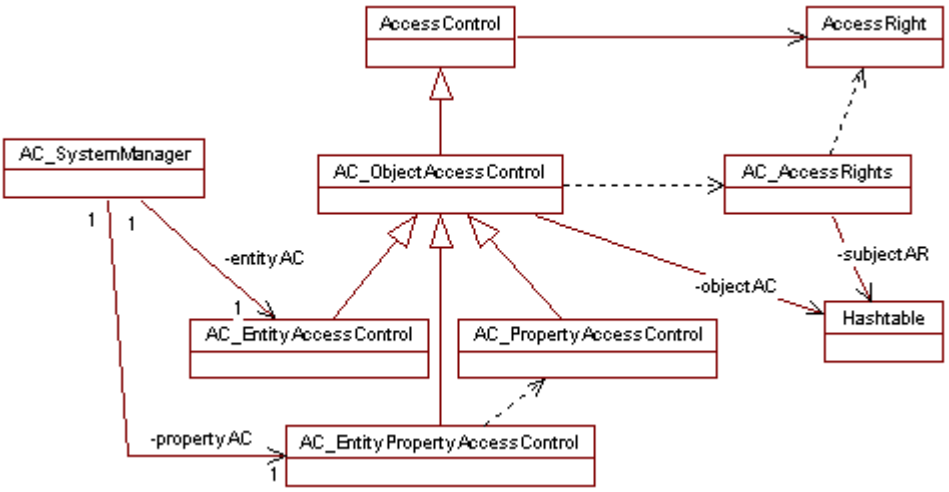


Figura 7.3: Diagrama de classes do módulo controle de acesso para o NCM.

A classe *AccessControl* foi especializada na classe *AC_ObjectAccessControl*, para definir a estrutura de armazenamento de direitos de acesso em tabelas (*hashtables*). Essa classe redefine os métodos de sua superclasse (*AccessControl*), para o armazenamento e consulta de diretos de acesso, que devem ser aplicados à nova estrutura.

A classe *AC_AccessRights* foi acrescentada ao modelo, apresentado na Figura 7.1, para compor a estrutura de armazenamento de direitos de acesso em tabelas. Essa classe possui uma *hashtable*, onde são armazenados os direitos de acesso (*AccessRight*) para cada sujeito com permissão sobre um objeto, e implementa os métodos para a manipulação desses direitos de acesso.

As classes *AC_EntityAccessControl, AC_EntityPropertyAccessControl*, *AC_PropertyAccessControl* foram definidas como especializações da classe *AC_ObjectAccessControl*. Essas classes implementam os métodos definidos na superclasse para entidades e propriedades NCM, de acordo com a lógica de tabelas definida para objetos.

A classe *AC_EntityAccessControl* armazena e regula os direitos de acesso definidos para *entidades NCM.* Para cada entrada (*entidade*) da tabela são armazenados objetos *AC_AccessRights*, representado os direitos de acesso de *usuários* sobre a *entidade*.

Para possibilitar a busca de direitos de acesso de propriedades, foram criadas as classes *AC_EntityPropertyAccessControl* e *AC_PropertyAccessControl*.

A classe *AC_PropertyAccessControl* representa os direitos de acesso definidos para *propriedades NCM.* Para cada entrada (*propriedade*) da tabela são armazenados objetos *AC_AccessRights*, representado os direitos de acesso de *usuários* sobre a *propriedade*. Como uma determinada *propriedade NCM* sempre representa um atributo de uma determinada *entidade NCM*, o direito de acesso de *propriedade* será armazenado, especificamente, como um direito de acesso de uma *propriedade* componente de uma determinada *entidade*. Para representar essa lógica foi criada a classe *AC_EntityPropertyAccessControl*.

AC_EntityPropertyAccessControl representa os direitos de acesso definidos para *propriedades NCM* pertencentes a *entidades NCM.* Essa classe define uma *hashtable*, onde, para cada entrada (*entidade*) da tabela, são armazenados objetos do tipo *AC_PropertyAccessControl*.

A classe *SystemManager* funciona como uma fachada para o módulo Controle de Acesso, e referencia os controles de acesso para *entidade* e *propriedade* (*entityAC* e *propertyAC*).

1.2 Módulo Controle de Permissão

Através desse módulo, serão definidos os *conjuntos de permissão* válidos para *entidades* e *propriedades*, de acordo com as *ações* que poderão ser realizadas sobre as mesmas. Nesse sentido, esse módulo, definido inicialmente para *entidades* em geral, poderá ser estendido para cada especialização de *entidade* do NCM, de acordo com as ações que poderão ser realizadas por cada uma dessas especializações.

O Controle de Permissão próprio a objetos NCM será especificado como uma especialização do seguinte diagrama.

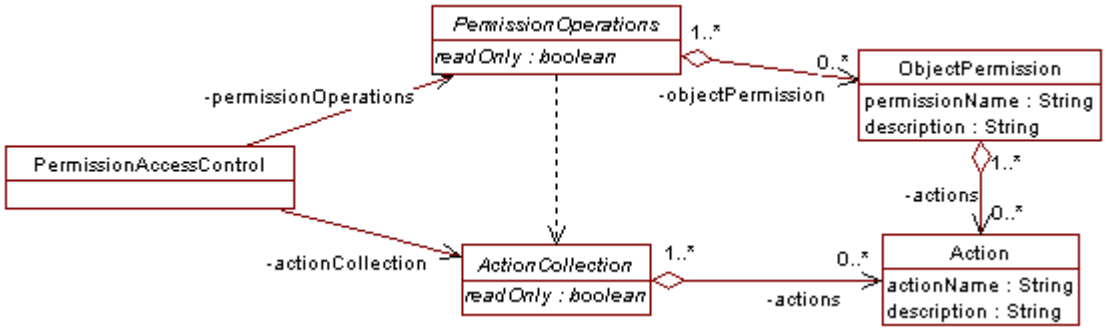


Figura 7.4: Diagrama de classes do módulo Permissão.

A classe *PermissionAccessControl* fornece a interface para o controle e manipulação de *permissões* e *ações* do sistema.

A classe *ObjectPermission* define o objeto *permissão*, e possui como atributos um nome, uma descrição e um conjunto de ações (*actions*). Essa classe define métodos para manipular seus atributos, como por exemplo, para verificar se uma determinada *ação* pode ser executada por um sujeito com uma certa *permissão*.

A classe *Action* representa uma *ação* possível de ser realizada sobre um *objeto*, especificando, basicamente, um método de um objeto que poderá ser executado. Essa classe define os atributos nome e descrição, onde o nome representa a assinatura de um método, e a descrição uma descrição do método.

A classe *ActionCollection* define um *conjunto de ações* que poderão ser realizadas sobre um *objeto* (ou tipo de objeto) por um *sujeito*. Essa classe fornece

métodos para criar e inserir uma nova ação[47], proibir a inserção de novas ações no conjunto de ações, e listar as *ações* desse conjunto.

O *conjunto de permissões*, que poderão ser atribuídas a um determinado *objeto*, é representado pela classe *PermissionOperations*. *Permissões*, pertencentes a um conjunto de permissões, só deverão conter ações pertencentes a um conjunto de ações, válidas para o *objeto* em questão[48] (esses conjuntos são relacionados para que permissões sejam definidas de maneira consistente). Essa classe fornece métodos para criar e inserir uma nova *permissão*[49], retornar as *ações* definidas em uma determinada *permissão*, proibir a inserção de novas permissões e listar as *permissões* desse conjunto.

As classes *PermissionAccessControl*, *PermissionsOperations* e *Action-Collection* representam pontos de flexibilização e devem ser instanciadas como se segue:

⇒ A instância da classe *ActionCollection* deve definir o *conjunto de ações* válidas para um determinado *objeto* (representando os métodos que poderão ser executados sobre um objeto).

⇒ A instância da classe *PermissionOperations* deve definir o *conjunto de permissões* para um *objeto*, conjunto esse que depende do *conjunto de ações* que podem ser utilizadas para compor uma *permissão*. Para cada instância dessa classe, deve ser definida uma instância da classe *ActionCollection*.

⇒ A classe *PermissionAccessControl* deve ser especificada de modo que sua instância determine os *conjuntos de permissões* e *ações* que poderão ser aplicados a um *objeto*. Para cada instância dessa classe devem ser definidas as instâncias de *PermissionOperations* e *ActionCollection*.

---

[47] Uma nova *ação* pode ser inserida no *conjunto de ações* de um *objeto*, se esse conjunto não for marcado como “read only”.

[48] A relação de dependência entre a classe *PermissionOperations* e *ActionCollection* representa essa restrição.

[49] Uma nova *permissão* pode ser inserida no *conjunto de permissões* de um *objeto*, se esse conjunto não for marcado como “read only”.

O pattern Factory Method [GAMMA] foi utilizado em *PermissionAccessControl* e *PermissionOperations*, quando foi deferida para as subclasses a instanciação necessária. O pattern define uma interface para a criação de objetos, mas deixa a cargo das subclasses qual classe instanciar.

O CONROLE DE PERMISSÃO PARA O NCM

Os pontos de flexibilização do Controle de Permissão foram instanciados para *entidades* e *propriedades NCM*, como se segue, sendo criados os conjuntos de permissões e ações correspondentes a essas entidades e propriedades.

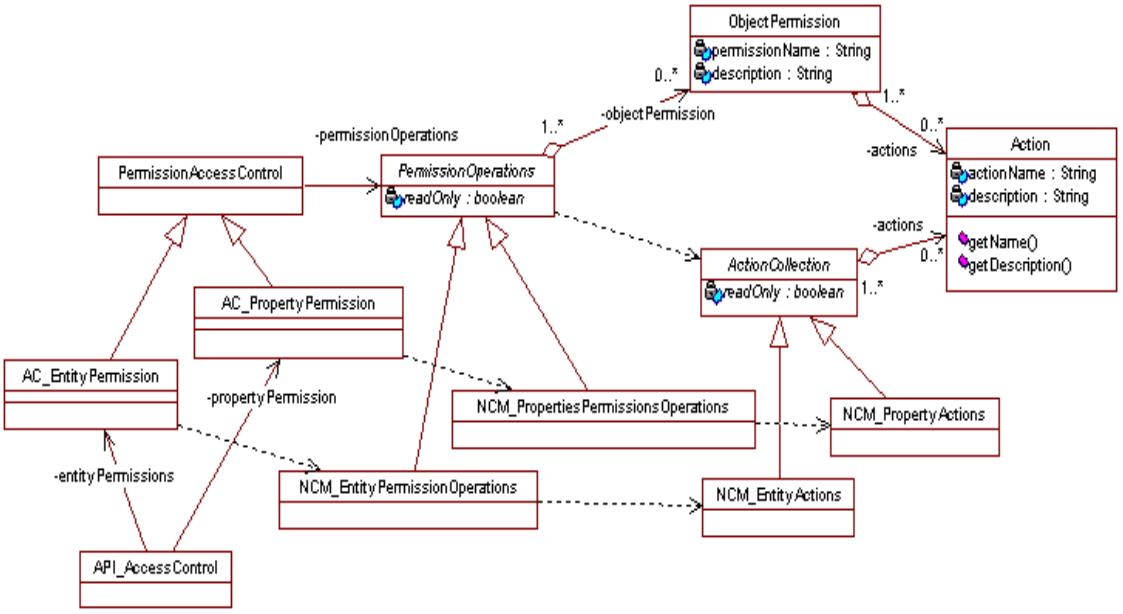


Figura 7.5: Diagrama de classes do módulo Permissão para o NCM.

Foram acrescentadas ao modelo, apresentado na figura anterior, as classes *AC_EntityPermission*, *AC_PropertyPermission*, *NCM_EntityPermissionOperations*, *NCM_PropertyPermissionOperations*, *NCM_EntityActions* e *NCM_PropertyActions*.

As classes *AC_EntityPermission* e *AC_PropertyPermission* são especializações da classe *PermissionAccessControl*, e definem os conjuntos de permissão que poderão ser aplicadas a entidades (*NCM_EntityPermissionOperations*) e propriedades (*NCM_PropertyPermissionOperations*), respectivamente. Esses conjuntos são instâncias da classe *PermissionOperation*, e serão definidos, por sua vez, de acordo com os

conjuntos de ações que poderão ser aplicadas a entidades (*NCM_EntityActions*) e propriedades (*NCM_PropertyActions*).

A classe *NCM_PropertyActions*, especialização da classe *ActionCollection*, define o seguinte conjunto de ações básicas que poderão ser aplicadas a *propriedades NCM*:

- *Read*: Permite a leitura de uma propriedade e de seu conteúdo;
- *Write*: Permite a alteração do valor de uma propriedade;
- *Delete*: Permite a remoção de uma propriedade. Esta ação é valida apenas para atributos estendidos[50] de uma entidade.

Ainda será definida uma ação *ChangePermission* que permite a alteração de uma determinada permissão para uma propriedade. Essa ação não retrata um método de uma entidade, mas sim um método do próprio controle de acesso.

Essas ações serão combinadas para a definição das seguintes *permissões*, que irão compor o conjunto de permissões definidas em *NCM_PropertyPermission-Operations*:

- *No Access*: Previne qualquer tipo de acesso à propriedade;
- *Read*: Permite acesso de leitura aos valores da propriedade, incluindo execuções;
- *Write*: Permite a leitura e alteração em valores de uma propriedade;
- *Change*: Permite a leitura e alteração em valores de uma propriedade. Esta permissão permite ainda a exclusão da propriedade, se for uma propriedade estendida;
- *Full Control*: Permite a leitura e alteração em valores de uma propriedade. Esta permissão permite ainda a exclusão da propriedade, se for uma propriedade estendida e a alteração da permissão de um usuário sobre uma propriedade.

[50] Entidades possuem uma lista de propriedades básicas, que representam seus atributos básicos, e uma lista de propriedades estendidas, que são atributos adicionados a entidades. Apenas propriedades estendidas podem ser removidas.

Outras permissões poderão ser criadas e atribuídas a sujeitos sobre propriedades, se forem permitidas inserções, desde que satisfaçam a restrição de conter apenas ações válidas a propriedades.

A classe *NCM_EntityActions*, especialização da classe *ActionCollection*, define o seguinte conjunto de ações básicas que poderão ser aplicadas a entidades:

- *List*: Lista as propriedades de uma entidade, sem, no entanto, mostrar seus valores;
- *Add*: Permite a inserção de novas propriedades na entidade (propriedades estendidas).

Ainda será definida uma ação *ChangePermission* que permite a alteração de uma determinada permissão para uma entidade. Essa ação não retrata um método de uma entidade, mas sim um método do próprio controle de acesso.

Essas ações serão combinadas para a definição das seguintes *permissões*, que irão compor o conjunto de permissões definidas em *NCM_EntityPermission-Operations*:

- *List*: Lista as propriedades de uma entidade, sem, no entanto, mostrar seus valores;
- *Add*: Permite a adição de novas propriedades na entidade sem alterar as permissões de propriedades já existentes;
- *Change*: Permite a visualização das propriedades de uma entidade e a adição de novas propriedades na entidade;
- *FullControl*: Permite a visualização das propriedades de uma entidade, a adição de novas propriedades na entidade, e ainda a alteração de direitos de acesso de usuários sobre a entidade.

Como para o conjunto de permissões sobre propriedades, outras permissões poderão ser criadas e atribuídas a sujeitos sobre entidades, se forem permitidas inserções, desde que satisfaçam a restrição de conter apenas ações válidas a entidades.

O controle de acesso, definido para entidades e propriedades, como foi dito anteriormente, poderá ser estendido de modo a definir novas permissões para diferentes entidades do modelo NCM, como deverá ser feito para bases privadas e documentos.

Nesse sentido, a classe *AC_EntityPermission* poderia ser estendida para formar as classes *AC_PrivateBasePermission* e *AC_DocumentPermission*, por exemplo.

A classe *API_AccessControl* foi criada para representar a interface com o *Controle de Acesso*, e é responsável pela comunicação entre o módulo controle de acesso, que define os direitos de acesso para ator*es* sobre *entidades* ou *propriedades*, e o módulo controle de permissão, responsável pela definição das *permissões* que serão utilizadas pelo Controle de Acesso.

Essa classe, ainda, servirá como mediador entre o Controle de Acesso e o sistema sobre o qual o controle irá atuar, tendo o papel de intermediar a chamada a métodos de *objetos* do sistema. Basicamente, a classe *API_AccessControl* deve verificar o direito de acesso de um determinado *sujeito*, que deseja realizar uma ação (executar um método) sobre um objeto, antes que o método seja efetivamente chamado pela própria classe mediadora, quando o *sujeito* tem permissão para realizar a ação.

2 Controle de Concorrência

O controle de concorrência baseado em travas especificado para o NCM deverá restringir o acesso concorrente a *objetos NCM* a apenas um *sujeito* por vez, mais especificamente, o controle de concorrência irá restringir um acesso concorrente a atores sobre *propriedades NCM.*

Quando todas as *propriedades* de uma *entidade* estiverem travadas, isto significará, por conseqüência, o bloqueio da *entidade* como um todo.

O controle de concorrência, próprio a objetos NCM, foi especificado seguindo o seguinte diagrama de classes.

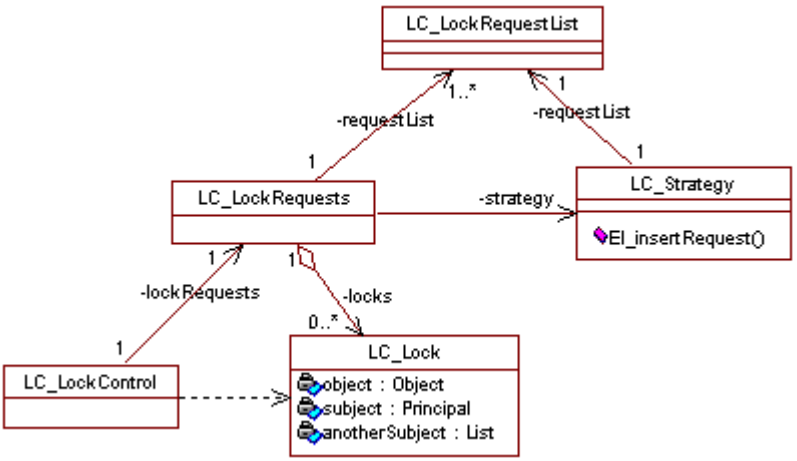


Figura 7.6: Diagrama de classes do controle de concorrência.

A classe *LC_LockControl* funciona como uma interface para o Controle de Travas, e é através dela que serão feitos:

- os pedidos de bloqueio de um objeto, por um determinado *sujeito*;
- os pedidos de desbloqueio de um *objeto*, por um determinado *sujeito*;
- uma consulta sobre *travas* em *objetos*; e,
- uma consulta sobre pedidos de *trava* em *objeto*.

Essa classe irá conter todas as travas (*LC_Lock*) ativas, atribuídas a um determinado sujeito sobre um determinado objeto.

Travas são representadas pela classe *LC_Lock*, que relaciona um *objeto*, sobre o qual o bloqueio será realizado, e um *usuário*, responsável pela trava. Essa classe ainda possui como outro atributo, uma *lista de usuários*, com permissão para destravar o objeto (quando, por exemplo, for necessário resolver um *deadlock*), e define os métodos para a manipulação de seus atributos.

A classe *LC_LockRequests* define uma fila de pedidos de trava (*LC_LockRequestList*), e métodos para manipular essa fila, inclusive para a inserção de pedidos de acordo com um estratégia (*strategy*).

A classe *LC_Strategy* define uma interface para a especificação de estratégias de inserção de pedidos de bloqueio de objetos na *fila de pedidos*, e deve ser especializada para que estratégias efetivamente sejam fornecidas.

CONTROLE DE CONCORRÊNCIA NO NCM

Como dito anteriormente, o Controle de Travas irá atuar em propriedades NCM. Para acrescentar agilidades na consulta de uma trava atribuída a um *usuário* sobre uma *propriedade* de uma determinada *entidade*, a estrutura de armazenamento de travas, definida em *LC_LockControl*, foi especializada em uma lógica de *hashtables*, muito semelhante àquela apresentada para direitos de acesso.

O Controle de Travas (*LC_LockControl*) irá armazenar, por entidade, as propriedades que estão travadas, e para cada propriedade, a trava relativa a um sujeito, conforme a figura seguinte.

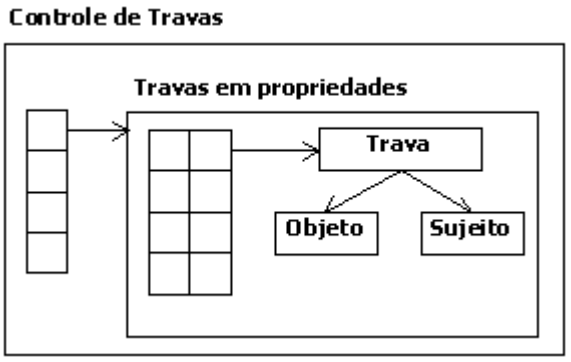


Figura 7.7: Estrutura de armazenamento de *travas*.

Definida essa estrutura, o diagrama de classes utilizado para implementar o Controle de Concorrência para o NCM pode ser visto a seguir.

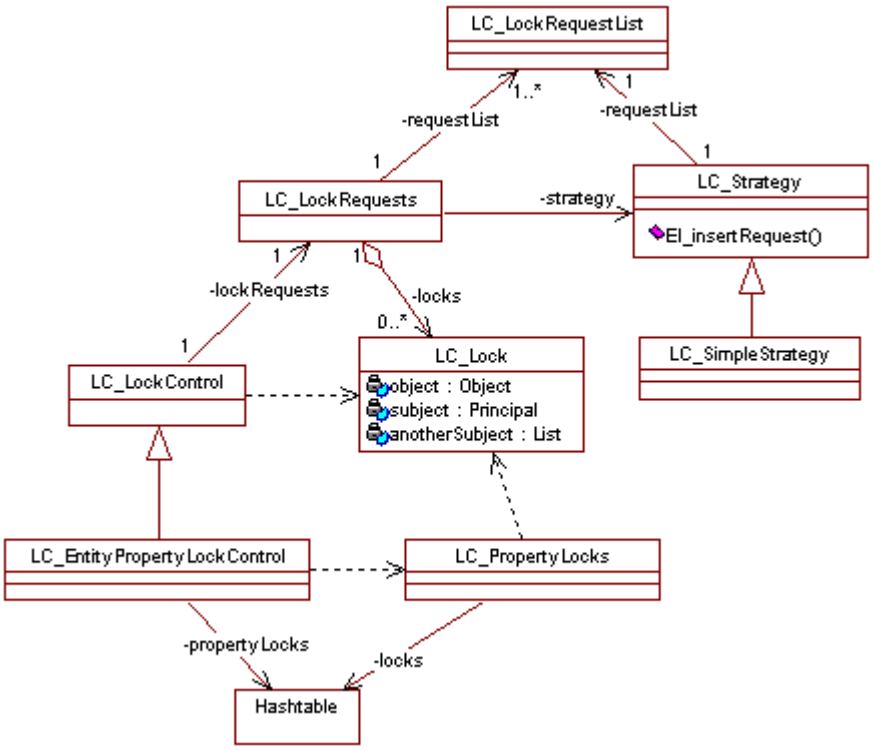


Figura 7.8: Diagrama de classes do controle de concorrência para o NCM.

As classes *LC_EntityPropertyLockControl* e *LC_PropertyLocks* foram acrescentadas ao diagrama de classes, apresentado na Figura 7.6, para implementar a lógica de armazenamento de travas por propriedades de entidades.

A classe *LC_EntityPropertyLockControl* contém uma *hashtable*, onde serão armazenadas as travas de suas propriedades. A classe *LC_PropertyLocks* representa esse conjunto de travas em propriedades de uma entidade e contém as travas (*LC_Lock*) propriamente ditas.

A classe *LC_SimpleStrategy*, definida como instância de *LC_Srategy*, define uma estratégia simples de inserção de travas da fila de pedidos (*LC_LockRequestList*), que apenas insere pedidos de travas por ordem de chegada.