Especificar a persistência dos modelos de objetos de sistemas distribuídos e adaptáveis aplicados ao controle de satélites.

Warley R. de Almeida
Instituto Nacional de Pesquisas Espaciais
warley@lac.inpe.br

Maurício G.V. Ferreira
Instituto Nacional de Pesquisas Espaciais
mauricio@ccs.inpe.br

Resumo

Alguns sistemas de informação, por seus domínios estarem se alterando constantemente, requisitam uma arquitetura que permita configurar e adaptar estes sistemas em tempo de execução. A estrutura e o comportamento dos objetos destes sistemas ao invés de codificados, ficam armazenados em um banco de dados, permitindo mudanças no modelo de objetos através de alterações neste banco de dados, sem a necessidade de recodificar o sistema. Este estilo arquitetural é chamado de Adaptive Object Model (AOM). Em um sistema desenvolvido de acordo com a arquitetura AOM, o modelo de objetos do domínio fica armazenado em um banco de dados e o que se codifica é um interpretador para este modelo de objetos. Uma das dificuldades apresentadas no desenvolvimento de um AOM é como armazenar e recuperar o modelo de objetos em um banco de dados. Em sistemas distribuídos desenvolvidos baseados na arquitetura AOM, a tarefa de persistir o modelo de objetos em um banco de dados, se apresenta como um novo desafio.

Palavras-chave: *Modelos de Objetos Adaptáveis, Reflexão, Padrões de Projeto, Metamodelagem e Persistência .*

1. Introdução

Alguns sistemas de informação, principalmente devido a características de seu domínio, podem ser desenvolvidos de uma forma que os tornem mais configuráveis e adaptáveis. Assim, estes sistemas podem se alterar de uma maneira mais eficiente para satisfazer requisitos do seu domínio de informação, que vierem a surgir (ou se modificar) após o período de desenvolvimento destes sistemas [12].

O Controle de satélites é um exemplo de domínio de informação que requisita um sistema configurável e adaptável. Cada satélite tem características próprias, o que significa que um satélite sempre difere de outro, mesmo que às vezes seja uma diferença sutil.

Atualmente, a cada lançamento de um novo satélite, deve-se desenvolver um aplicativo específico para este satélite e destinar uma máquina ou um conjunto máquinas específicas, para a execução deste aplicativo, auxiliando no recebimento de seus dados e no monitoramento de seu estado interno.

Estas características, existentes em determinados domínios de informação, aliadas ao desejo crescente de se obter aplicações que evoluam à medida que seus domínios de informação evoluam, fez com que se pensasse em construir sistemas para o controle de satélites mais configuráveis, flexíveis e adaptáveis, permitindo que o sistema possa se adaptar mais facilmente, às novas necessidades do domínio de informação [9].

Uma maneira de se conseguir aplicações mais configuráveis é armazenar certos aspectos do sistema, como as regras do negócio por exemplo, em um banco de dados, permitindo que se altere estas regras de negócio, sem a necessidade de se alterar o código que as implementam. O modelo resultante, permite que o sistema possa se adaptar mais facilmente às novas necessidades do domínio, através de modificações nos valores armazenados em um banco de dados, ao invés de alterações no código que implementa o sistema.

Uma arquitetura de modelos de objetos adaptáveis (*Adaptive Object Model Architecture*) é um tipo particular de arquitetura reflexiva que abrange sistemas orientados a objeto que gerenciam elementos de algum tipo, e que podem ser estendidos para adicionar novos elementos (adaptável) ou modificar elementos existentes (configurável). Sistemas que têm este tipo de arquitetura recebem também os nomes de Modelos de Objetos Ativos (*Active Object Models*) ou Modelos de Objetos Dinâmicos (*Dynamic Object Models*) [12].

Uma abordagem dos modelos de objetos adaptáveis no desenvolvimento de sistemas pode amenizar algumas das dificuldades encontradas atualmente pelos desenvolvedores de software, principalmente em relação à flexibilidade, evolução e manutenção do sistema desenvolvido, permitindo uma redução do custo total tanto no desenvolvimento quanto na manutenção destes sistemas [7].

Por exemplo, a partir do momento em que se consiga implementar um sistema adaptável para o controle de satélites com a capacidade de: (1) armazenar o modelo de objetos do sistema em um banco de dados, (2) permitir ao usuário monitorar qualquer um dos satélites em um determinado instante, (3) manter o modelo de objetos e (4) adaptar o modelo de objetos a um novo satélite. Este sistema ajudará o programa espacial brasileiro na operação de satélites, pois terá mais flexibilidade para atender novas missões.

Com a implementação deste sistema adaptável, o esforço necessário para se disponibilizar softwares para o controle de novos satélites tende a diminuir. Isto se deve ao fato de que adaptar um sistema configurável e adaptável para o controle de

satélites à um novo satélite, pode ser realizado em menos tempo e a um custo mais baixo do que desenvolver um novo sistema para o controle deste novo satélite.

Um sistema baseado na arquitetura AOM fornece a seus usuários, informações sobre o seu próprio modelo, permitindo assim, que seus usuários possam alterar o modelo do sistema, de acordo com suas necessidades, em tempo de execução, através da modificação destas informações. Esta característica, torna o sistema mais configurável e adaptável, em contrapartida, este tipo de arquitetura também apresenta a dificuldade de armazenar este modelo, pois um sistema desenvolvido baseado na arquitetura AOM, armazena além dos valores dos atributos de um objeto, armazena também as estruturas destes atributos e ainda armazena os próprios métodos executados por este objeto.

Projetar a persistência dos objetos de um sistema é uma tarefa difícil e que consome tempo. A qualidade estrutural de um banco de dados, depende sobretudo da metodologia de projeto usada e da técnica de projeto usada nos passos desta metodologia.

O projeto da persistência dos objetos de um sistema baseado na arquitetura AOM, acrescenta algumas dificuldades, principalmente devido ao fato destes sistemas armazenarem além dos valores dos atributos dos objetos, armazenarem os próprios atributos e métodos destes objetos.

Estas dificuldades foram uma motivação para este trabalho, cujo objetivo principal é estabelecer métodos e definir um mapeamento para a persistência dos modelos de sistemas desenvolvidos baseados na arquitetura AOM. O resultado principal é dar aos desenvolvedores destes sistemas, um maior entendimento da arquitetura e um embasamento que os ajudem a desenvolver sistemas mais eficientes e em um menor espaço de tempo.

Este trabalho tem como estudo de caso, para o mapeamento criado, a proposta de Thomé [9] de uma arquitetura de software distribuída configurável e adaptável aplicada a várias missões de controle de satélite.

2. Modelo de Objetos Adaptáveis

Um Modelo de Objetos Adaptável (*Adaptive Object Model* - AOM) é um sistema que apresenta informações sobre suas classes, atributos e associações como metadados (Um metadado é um dado que descreve outro dado. Um metadado descreve a estrutura e o significado deste outro dado [2]). O modelo do sistema é baseado nas instâncias das classes ao invés das classes. Usuários especialistas no domínio (um usuário com conhecimento sobre desenvolvimento de sistemas, ou apoiado por uma pessoa ou equipe de desenvolvimento de sistemas) modificam os metadados (modelo de objetos) para refletir mudanças no domínio. Estas mudanças modificam a estrutura (atributos de um objeto) e o comportamento (métodos de um objeto) do sistema. Em outras palavras, o AOM armazena seu modelo de objetos em um banco de dados e em tempo de execução o interpreta. Conseqüentemente, o modelo de objetos é adaptável, pois quando você o altera, o sistema se adapta imediatamente refletindo a mudança realizada [12].

Desta forma, os metadados são usados em modelos de objetos adaptáveis para descrever o modelo em si (auto-representação, portanto uma arquitetura reflexiva). Desde que o sistema consiga interpretar os metadados para construir e manipular as descrições das classes do modelo em tempo de execução, simplifica-se o processo de adicionar novas classes ao modelo de objetos adaptável. Simplifica-se também o processo de alterar as classes já existentes no modelo, e torná-las imediatamente disponíveis para os usuários do sistema[2].

O projeto de um sistema baseado na arquitetura AOM envolve três atividades principais: (1) definir as entidades, regras e associações do negócio; (2) desenvolver mecanismos para instanciação e manipulação destas entidades de acordo com suas regras e associações; e (3) desenvolver ferramentas que permitam aos usuários especialistas no domínio modificarem a descrição destas entidades, regras e associações (representados como metadados) de acordo com as necessidades do domínio [12].

Essas atividades podem ser realizadas através da aplicação de um ou mais padrões de projeto, em conjunto com regras de engenharia de software para análise, projeto e implementação de sistemas. Os padrões de projeto *Type Object* [6], *Property* [4], *Strategy* [4], o *Rule Object* [1] e o *Composite* [4], são freqüentemente empregados na a construção de modelos de objetos adaptáveis [12].

Os aspectos primários de implementação que precisam ser abordados no desenvolvimento de sistemas que utilizem modelos de objetos adaptáveis são: (1) como armazenar o modelo de objetos no banco de dados, (2) como apresentar o modelo de objetos para o usuário, (3) como dar ao usuário especialista no domínio, condições de realizar alterações no modelo de objetos e (4) como lidar com o histórico das alterações realizadas pelo usuário especialista no domínio no modelo de objetos [11].

Os AOMs representam seus objetos como metadados, isto significa que seus modelos de objetos podem ser armazenados, como metadados, em bancos de dados orientado a objetos, bancos de dados relacionais ou em arquivos XML. Independente da maneira escolhida para o armazenamento dos metadados, o sistema deve ter a capacidade de ler os metadados e instanciar o modelo de objetos adaptável com a configuração correta de suas instâncias. Isto pode ser conseguido usando-se construtores (utilizando o padrão de projeto *Builder* [4], por exemplo) e interpretadores dos metadados para a criação e configuração das entidades, atributos, associações e comportamentos a partir dos metadados.

O modelo de objetos adaptável pode ficar armazenado tanto em um banco de dados, quanto em arquivos XML. Independentemente da forma como o modelo de objetos for armazenado, ele deve ser instanciado e ficar disponível no repositório de metadados para que possa ser utilizado pela aplicação. No caso de se utilizar um banco de dados para armazenar o modelo de objetos, o mecanismo de persistência e o interpretador de metadados realizam estas tarefas; e no caso do modelo de objetos ficar armazenado em arquivos XML, o XML *Parser* e o interpretador de metadados realizam estas tarefas. Se houver a necessidade de persistir os objetos do

domínio da aplicação, eles podem ser armazenados no banco de dados através do mecanismo de persistência.

Alterações realizadas no modelo de objetos refletem diretamente no comportamento do software, resultando em uma extensão completa em tempo de execução. Por exemplo, alterar uma classe do sistema e armazenar a nova versão desta classe faz com que todos os sistemas de software no ambiente, automaticamente utilizem a nova versão desta classe. Assim, pode-se atualizar os metadados durante a execução do sistema, e as mudanças resultantes no comportamento e estrutura do sistema têm efeito assim que o modelo de objetos é interpretado [8].

Para o armazenar do modelo de objetos, como metadados, em um banco de dados ou em arquivos XML, os desenvolvedores terão que definir como representar o modelo, bem como a semântica para descrever as regras do negócio. Deve-se estabelecer um mapeamento para a persistência do modelo de objetos em um banco de dados, ou em arquivos XML. Além disto, deve-se desenvolver editores de metadados e ferramentas de programação para auxiliar os usuários especialistas no domínio nas tarefas de criação e manutenção do modelo de objetos[11].

Uma das principais razões para se desenvolver um sistema baseado nos modelos de objetos adaptável é permitir que usuários especialistas no domínio possam modificar a estrutura e o comportamento do sistema definindo novas entidades, associações e regras ou alterando as já existentes, sem a necessidade de modificar o código do sistema. Para que os usuários especialistas no domínio possam realizar estas modificações, com uma menor probabilidade de introdução de erros no sistema e de uma maneira mais intuitiva, é preciso oferecer a estes usuários, ferramentas adequadas para que eles possam realizar estas tarefas de uma maneira mais eficiente. Conseqüentemente, é essencial para um sistema adaptável, possuir ferramentas para a definição de novas entidades, associações e regras de forma interativa e ferramentas para apresentar o modelo de objetos a estes usuários, permitindo a modificação de entidades, associações e regras existentes [12].

Para este trabalho foi desenvolvido três protótipos de um sistema baseado nos AOMs, tendo como origem a proposta de Thomé [9] para o controle de satélites. O primeiro protótipo persiste o modelo de objetos em um SGBDOO (Sistema Gerenciador de Banco de Dados Orientado a Objeto), o segundo em um SGBDR (SGBD Relacional) e o terceiro armazena o modelo de objetos em arquivos XML em um banco de dados de XML nativo. No desenvolvimento destes protótipos foi implementado editores de metadados, para a criação de entidades, de propriedades e de regras (métodos) que podem ser associados às entidades criadas pelos usuários especialistas no domínio do sistema. Os três protótipos foram desenvolvidos na linguagem Java e seguindo a arquitetura J2EE.

A linguagem Java pode ser considerada muito mais do que apenas uma linguagem de programação. Java consiste também de um ambiente de desenvolvimento e execução de sistemas, com características marcantes como interoperabilidade, independência de plataforma e Orientação a Objetos. A linguagem Java oferece uma série de APIs (*Application Program Interface*) que são interfaces entre a aplicação e a plataforma que a processa [5].

A linguagem Java é apropriada para a implementação de sistemas baseados na arquitetura AOM por oferecer suporte a vários tipos de dados utilizados no desenvolvimento destes sistemas (Exemplos: As classes *Vector*, *HashTable* e *Class*), fornece também APIs para a manipulação de arquivos XML (Exemplos: JMI, XMLDecoder, XMLEncoder), além de APIs para conexões com diversos SGBDs relacionais e orientados a objetos (JDBC). A linguagem Java também possui a API *Reflection*, utilizada para o desenvolvimento de aplicações dinâmicas. A API *Reflection* fornece suporte para a execução dinâmica de métodos [5].

A reflexão habilita uma aplicação descobrir informações sobre os campos, métodos e construtores de objetos em tempo de execução, e permite também refletir em um outro objeto, os campos, métodos e construtores de um determinado objeto, tornando possível a sua execução. Você pode necessitar da API de reflexão em Java, quando estiver trabalhando no desenvolvimento de ferramentas como *debugers*, *browsers* de classes e construtores de GUIs (*Graphical User Interface*) [5].

Atualmente, uma das arquiteturas utilizadas para o desenvolvimento de sistemas distribuídos é a arquitetura J2EE (*Java 2 Enterprise Edition*). A arquitetura J2EE permite aos desenvolvedores a criação de robustas aplicações em três camadas (também chamada de N camadas), através do fornecimento de serviços da camada *middleware* para comunicar com uma variedade de clientes e serviços *back end* (Notadamente os servidores de banco de dados). A arquitetura J2EE, fornecerá o suporte necessário para a distribuição dos objetos da aplicação, facilitando assim a implementação dos protótipos para a edição dos metadados que representam os modelos de objetos [3].

3. Persistindo os Modelos de Objetos Adaptáveis

Uma das maneiras de facilitar a persistência e a instanciação dos modelos de objetos adaptáveis, é a criação de um mapeamento. Um mapeamento pode ser entendido como um conjunto de regras a serem seguidas para se obter um resultado. Pode-se criar um mapeamento para a persistência do modelos de objetos em banco de dados e um outro mapeamento para a persistência do modelo de objetos em arquivos XML. Para formalizar um mapeamento, pode-se utilizar mecanismos de extensão da UML, como por exemplo os estereótipos, na criação dos modelos de objetos. Assim cria-se estereótipos específicos para representar o modelo adaptável, e adicionar as semânticas necessárias para a sua persistência [10].

O desenvolvimento de um sistema baseado na arquitetura dos AOMs, implica na criação de um metamodelo, que irá definir as regras para a criação de modelos específicos para o domínio, em tempo de execução. Para facilitar a formalização do mapeamento da persistência dos modelos de objetos adaptáveis criados em tempo de execução, definiu-se alguns estereótipos, que são inseridos nas classes do metamodelo, indicando qual o papel destas classes nos padrões de projeto utilizados. A tabela 1 mostra estes estereótipos e suas descrições.

Tabela 1 – Estereótipos para o metamodelo.

Estereótipo	Descrição
«Entidade»	Representa as entidades do negócio.
«TipoEntidade»	Representa os tipos de entidades do negócio.
«Propriedade»	Representa as propriedades de uma entidade do negócio.
«TipoPropriedade»	Representa os tipos de propriedades que uma entidade do negócio pode ter.
«TipoRelacionamento»	Representa os tipos de associações que podem existir entre as tipos de entidades do negócio.
«Relacionamento»	Representa as associações entre as entidades do negócio.
«Regra»	Representa a classe que armazena as regras que podem ser associadas as entidades do negócio.

Uma vez criado o metamodelo do sistema, deve-se adicionar ao metamodelo os estereótipos corretamente. Um estereótipo em uma classe, indica qual padrão de projeto foi utilizado para a criação desta classe no metamodelo (além da função desta classe no padrão de projeto) e para cada classe, o desenvolvedor deve seguir o mapeamento criado para a sua persistência, de acordo com o gerenciador de dados escolhido (Banco de dados orientados a objeto, banco de dados relacional ou arquivos XML) [3].

3.1. Persistindo as propriedades

Para persistir as propriedades do Modelo de Objetos, precisamos estabelecer um mapeamento para persistir as classes criadas pela aplicação dos padrões de projeto *Type Object* e *Property*. Os tipos das propriedades, podem ser armazenados como *strings*, e durante a execução do sistema, as propriedades podem ser validadas pela API *reflection* da linguagem Java, especialmente pelo método *forName* da classe *Class*.

A persistência das propriedades, compreende a persistência das classes marcadas com os estereótipos «Propriedade» e «TipoPropriedade»,

A classe TipoPropriedade estará associada a todas as classes marcadas com o estereótipo «TipoEntidade» do metamodelo, então estas associações devem ser tratadas no mapeamento.

Uma das formas de se tratar as associações entre a classe do metamodelo com o estereótipo «TipoPropriedade» e as classes do metamodelo marcadas com o estereótipo «TipoEntidade» é criar na classe «TipoPropriedade» uma referência para cada classe marcada com o estereótipo «TipoEntidade» no metamodelo. Desta forma um tipo de propriedade sempre vai pertencer a apenas um tipo de entidade.

Outra maneira de realizar a persistência das propriedades é considerar que um tipo de propriedade pode ser utilizado por vários tipos de entidade. Desta forma, as classes marcadas com o estereótipo «TipoEntidade» devem possuir uma lista ou um vetor de referências para instâncias da classe marcada com o estereótipo «TipoPropriedade» do metamodelo.

Esta segunda forma de persistir as propriedades parece mais condizente com os padrões de projeto *Type Object* e *Property*, onde a classe TipoPropriedade, se apresenta como um banco de propriedades, onde as classes marcadas com o estereótipo «TipoEntidade» possuiriam um subconjunto das instâncias destas classes. Inclusive a propriedade Nome da classe TipoPropriedade nesta abordagem seria único o que não ocorre na primeira abordagem.

Persistir as Propriedades em um banco de dados orientado a objeto, segundo a primeira abordagem citada anteriormente, consiste na criação duas classes: (1) TipoPropriedade com os atributos do tipo *string*: Nome e Tipo, além de uma referência para cada classe do metamodelo marcada com o estereótipo «TipoEntidade». (2) Propriedade com um atributo do tipo *string* Valor, uma referência para a Classe TipoPropriedade e mais uma referência para cada classe marcada com o estereótipo «Entidade» existente no metamodelo.

3.2. Persistindo as Associações

As associações podem ser consideradas como atributos de uma classe cujo valor seja outra classe. Além disto, as associações são bidirecionais, por exemplo se a classe a está associada com a classe B, então a classe B também está associada com a classe A. Para permitir que o usuário especialista no domínio, crie e altere associações entre as classes do metamodelo marcadas com o estereótipo «TipoEntidade», sem a necessidade de alterar o código do sistema, deve-se armazenar estas associações no dispositivo de persistência utilizado.

Persistir as associações, compreende persistir as classes do metamodelo marcadas com os estereótipos «Relacionamento» e «TipoRelacionamento», mostrados na tabela 1.

Além do fato das associações serem bidirecionais, cada classe pertencente a associação, possuiu uma cardinalidade diferente. Por isto, uma associação, será persistida como duas instâncias, uma para cada sentido da associação.

Persistir as associações em um banco de dados orientado a objetos, consiste na criação de duas classes, a classe TipoRelacionamento e a classe Relacionamento. A classe TipoRelacionamento irá conter as informações sobre os tipos de associações (com outra classe) que uma determinada classe marcada com o estereótipo «TipoEntidade» pode participar. Então a classe TipoRelacionamento deve armazenar informações sobre a multiplicidade desta classe na associação e também a referência da outra classe na associação. A Classe Relacionamento deve conter a instância da classe TipoRelacionamento a qual está associada, e irá associar instâncias das classes marcadas com o estereótipo «Entidade».

A classe TipoRelacionamento, possuirá os atributos Multiplicidade1 e Multiplicidade2, o atributo Nome, mais dois atributos (para armazenar referências) para cada classe marcada com o estereótipo «TipoEntidade» existente no metamodelo. O nome destes atributos podem ser o nome da classe marcada com o estereótipo «TipoEntidade» acrescido dos sufixos 1 ou 2.

A classe Relacionamento possuirá um atributo para armazenar uma referência ao seu TipoRelacionamento mais dois atributos (para armazenar referências) para cada classe marcada com o estereótipo «Entidade» existente no metamodelo. O nome destes atributos podem ser o nome da classe marcada com o estereótipo «Entidade» acrescido dos sufixos 1 ou 2.

3.3. Persistindo as Regras

O desenvolvimento de um sistema baseado nos AOMs será fortemente ditado pelas características do domínio de informação ao qual este sistema se destina. Obter um grande dinamismo nas regras do negócio do domínio, pode significar a criação de uma linguagem de alto nível específica para o domínio, o que implica criação de gramáticas, *parsers* e compiladores. Isto seria de uma complexidade maior, saindo do escopo deste trabalho. Se o domínio de informação para o qual se desenvolve um sistema baseado nos AOMs não requisitar um amplo dinamismo nas regras do negócio, certamente não compensará o esforço despendido para obter tal característica.

Neste trabalho, apenas a utilização do padrão de projeto *Strategy* foi formalizada no mapeamento e implementada nos protótipos. A utilização do padrão de projeto *Rule Object* fica para um trabalho futuro. Os protótipos desenvolvidos, baseados no padrão de projeto *Strategy*, implementaram para cada método disponível uma série de variações, e forneceu formas de o usuário especialista no domínio associar estes métodos a alguma classe marcada com o estereótipo «Entidade» e escolher qual das variações do método esta classe irá utilizar.

Persistir as regras do modelo, consiste em persistir a classe marcada com o estereótipo «Regra» e suas associações com as classes marcadas com o estereótipo «TipoEntidade» e com o estereótipo «TipoPropriedade».

Criou-se a classe Regra, que armazena todas as regras do negócio disponíveis, o único atributo desta classe é o atributo Nome. O protótipo fornece condições ao usuário especialista no domínio de associar estas regras às instâncias criadas das classes com o estereótipo «TipoEntidade». A associação entre a classe Regra e as classes com o estereótipo «TipoEntidade» no metamodelo é M:N.

Com o intuito de aumentar o dinamismo na execução dos métodos, criou-se a possibilidade de associar à uma determinada regra, um conjunto de propriedades. As Regras podem utilizar os valores destas propriedades tanto para indicar qual variação da regra deve ser utilizada, quanto para sua própria execução.

Durante a execução de uma regra, o sistema deve localizar se existe alguma propriedade compartilhada para esta regra. Se a regra possui alguma propriedade compartilhada o sistema deve buscar o seu valor na classe Propriedade, e fornecê-lo a regra antes de sua execução.

Desta forma, pode-se por exemplo uma associar uma regra a uma classe marcada com o estereótipo «TipoEntidade». Em seguida, pode se determinar um dos tipos de propriedades definidos para esta classe para que a regra utilize esta propriedade na sua execução.

Na implementação das regras no protótipo foi utilizada a API *Reflection* da linguagem Java, citada na seção 2 deste trabalho. Esta API fornece condições de localizar classes e executar métodos, cujos nomes só se tornem conhecidos em tempo de execução. Exatamente o que ocorre em sistemas desenvolvidos baseados nos AOMs.

Nos protótipos foi implementado em Java uma classe chamada Regras, que cuida da execução dos métodos, de acordo com o nome do método. O nome do método a ser executado fica persistido, em um dos dispositivos de persistência escolhido, na classe Regra. Durante a execução de um método, os protótipos verificam se existem propriedades compartilhadas para a execução deste método. Caso existam, os protótipos buscam os valores destas propriedades compartilhadas e fornecem ao método que será executado.

A figura 1 ilustra o método mais importante da classe Regras, chamado executarRegra. Este método possui como parâmetro o nome da regra, e os possíveis parâmetros com seus respectivos tipos (estes parâmetros são as propriedades compartilhadas pela classe Regra e as classes com o estereótipo «TipoEntidade»). A primeira linha recupera um objeto *Class* para a classe Regras, onde estão codificados todos os métodos. A segunda linha recupera o método desejado, a terceira linha cria uma instância da classe regra, necessária como parâmetro na quarta linha. Finalmente a quarta linha executa o método.

```
public static void executarRegra(String nomeregra,
                                 Class partType[], Object arglist[]){
  try {
    Class cls = Class.forName("User.Regras");
    Method meth = cls.getMethod(nomeregra, partType);
    Regras regra = new Regras();
    Object retobj = meth.invoke(regra, arglist);
  } catch (Throwable e ) {
    System.err.println(e);
  }
}
```

Figura 1 – Listagem do método executarRegra.

3.4. O protótipo

O protótipo desenvolvido, que exemplifica a capacidade que um sistema desenvolvido baseado nos AOMs, fornece aos seus usuários de configurar e adaptar o seu sistema a mudanças no domínio de informação. Foi desenvolvido editores de metadados, onde o usuário especialista do domínio pode criar novas classes, definir os atributos destas classes, criar associações entre estas classes e atribuir regras a estas novas classes.

Como todas estas informações são persistidas, na sua inicialização, o protótipo busca estas informações e instancia o modelo de objetos com a configuração definida pelo usuário especialista no domínio. A Figura 2 ilustra uma tela do protótipo mostrando duas classes diferentes criadas pelo usuário especialista no domínio em tempo de execução e uma tela onde o usuário especialista no domínio pode definir novos atributos

para uma classe. Maiores detalhes sobre o domínio de informação do protótipo podem ser encontrados em [9].

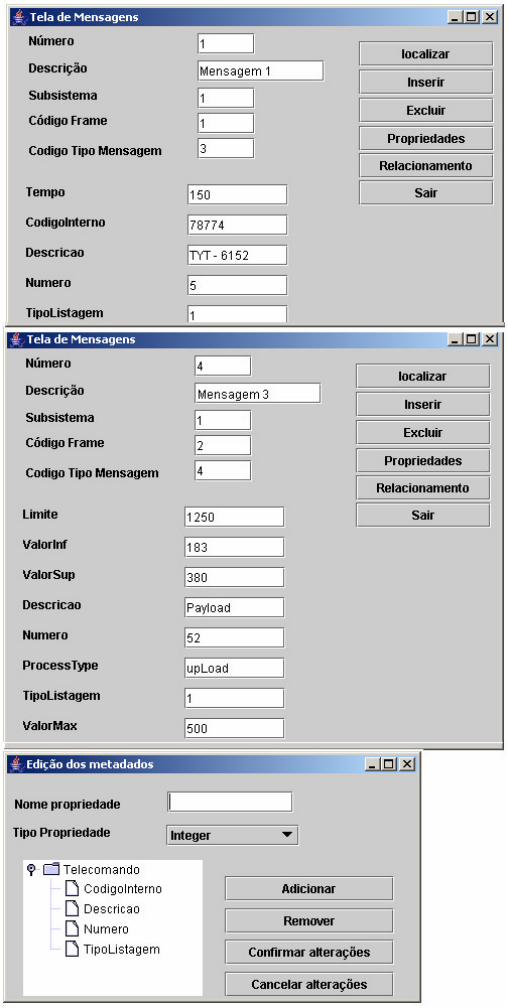

Figura 2 – Telas do protótipo desenvolvido.

4. Conclusões

O principal objetivo deste trabalho é estabelecer um mapeamento para a persistência dos modelos de objetos de sistemas distribuídos, configuráveis e adaptáveis, desenvolvidos baseados na arquitetura dos AOMs. Este trabalho, tem como origem, motivação e estudo de caso a arquitetura proposta em [9] para o controle de satélites.

Este trabalho utiliza três dispositivos de persistência, SGBDs Relacionais, SGBDs Orientados a Objetos e arquivos XML. Para isto, criou-se um mapeamento para a persistência dos modelos de objetos do sistema em cada um desses dispositivos. O protótipo implementou editores de metadados dos modelos de objetos, para facilitar a sua manutenção pelos usuários especialistas no domínio.

Independente da maneira com qual os metadados que descrevem o modelo de objetos, fiquem armazenados, o sistema deve ter a capacidade de ler estes metadados e instanciar o modelo de objetos adaptável correspondente. Além de manter a configuração correta da estrutura e comportamento deste modelo.

Ao término deste trabalho, espera-se colaborar para facilitar o desenvolvimento de sistemas distribuídos, configuráveis e adaptáveis para uso geral, mas utilizando como estudo de caso e focando principalmente em um sistema para o controle de satélites. Estes sistemas são um importante passo na direção da economicidade e reusabilidade, pois futuras missões de satélites poderão aproveitar todo o investimento de hardware e software já feito em missões anteriores. A contribuição principal é fornecer um embasamento para a persistência dos modelos de objetos destes sistemas. Diminuindo assim, o tempo e o esforço necessário para atender os requisitos de software das novas missões espaciais e contribuindo para o sucesso do Programa Espacial Brasileiro.

5. Referências

[1] ARSANJANI, A. **Rule Object: A Pattern Language for Adaptive and Scalable Business Rule Construction**. In: Conference on Patterns Languages of Programs (PloP'2000). Proceedings. Washington University Departament of Computer Science. October of 2000.

[2] FOOTE, B.;YODER, J.W. **Metadata and Active Object Models.** In: Technical Report#WUCS-97-34 (PLoP'98). Dept. of Computer Cience, Washington University, 1998.

[3] GABRICK, A. K.; WEISS, D. B. **J2EE and XML development.** Manning publications company, April 2002.

[4] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995.

[5] JAVA TUTORIAL. Reflection. Disponível em: <http://java.sun.com/docs/books/tutorial/index.html>. Acessado em 10/09/2004.

[6] JOHNSON, R.; WOOLF, B. **Type Object**. In: Pattern Languages of Program Design 3. Addison-Wesley, 1998. Page 47-66.

[7] LEDECZI, A.; KARSAI, G.; BAPTY, T. **Synthesis of self-adaptive software**. IEEE Aerospace Conference Proceedings. USA. V 4, pg. 501-507. 2000.

[8] POOLE, J. D. **Model-Driven Architecture: Vision, Standards And Emerging Technologies**. In: ECOOP´2001 Workshop on Metamodeling and Adaptive Object Models. April 2001.

[9] THOMÉ, A. C.; FERREIRA, M.G.V.; CUNHA, J.B.S. **Uma Arquitetura de Software Reflexiva Baseada em Modelos de Objetos Adaptáveis**. In: Object-Oriented Programming, System, Languages, and Applications (OOPSLA'04), Vancouver, Canadá, Outubro, 2004.

[10] WITTHAWASKUL W.; JOHNSON, R. **Specifying Persistence in Platform Independent Models**. In: Workshop in Software Model Engineering at The Sixth International Conference on the Unified Modeling Language, UML 2003, San Francisco, California, USA October 20-24, 2003.

[11] YODER, J.W.; JOHNSON R. **The Adaptive Object-Model Architectural Style**. In: IEEE/IFIP Conference on Software Architecture 2002 (WICSA3'02) Canada. August of 2002.

[12] YODER, J.W.; BALAGUER F.; JOHNSON R. **Architecture and design of Adaptive Object Models**. ACM Sigplan Notices. Vol 36, Fasc. 12, pg. 50-60, December 2001.