UMA ANÁLISE DA OMG MODEL DRIVEN ARCHITECTURE

Peter Jandl Junior[*]

RESUMO

Este trabalho discute a model driven architecture, sua concepção geral, a forma proposta de sua utilização e os benefícios esperados de sua aplicação no processo de desenvolvimento de software, tomando o cenário tecnológico atual das aplicações corporativas. Também efetua a contraposição das vantagens declaradas com alguns dos problemas envolvidos, realizando um balanço dos pontos positivos e negativos encontrados.

Palavras-chave: arquitetura de software, engenharia de software, MDA, model driven architecture, modelagem, UML.

ABSTRACT

This paper discusses the Model Driven Architecture, its general conception, the proposal for its use and the expected benefits of its employment in the software development process, in the present technological field of corporate applications. The paper brings a comparison between the declared advantages and some of its problems, taking into account the positive and negative points that have been found.

Key words: MDA – Model Driven Architecture; modeling; software architecture; software engineering; UML.

INTRODUÇÃO

O desenvolvimento e manutenção de aplicações dentro do contexto empresarial é uma tarefa árdua e complexa (HART, 2003; MILLER; MUKERJI, 2001; MILLER et al., 2003; SIEGEL, 2002; SOLEY, 2000). Com o passar do tempo as necessidades das organizações se modificam como decorrência natural do crescimento do negócio, da fusão ou associação com outras empresas, da concorrência, da evolução tecnológica e também de outros fatores relacionados ao mercado, à sociedade ou mesmo à legislação vigente. As necessidades de uso de informações mantidas por diferentes sistemas, projetados para atender necessidades temporárias ou muito específicas, eventualmente acabam por exigir sua adaptação para que sejam integrados entre si ou como forma de atender novas demandas.

Independente da aquisição de soluções prontas oferecidas pelo mercado ou mesmo do desenvolvimento de novos sistemas proprietários, a mudança natural do

[*] Mestre em Educação (USF) e Engenheiro de Eletrônica (Unicamp). Professor de cursos de Graduação, Pós-Graduação e Extensão da Universidade São Francisco e do Centro Universitário Padre Anchieta.

cenário das empresas gerará novas necessidades. Como a substituição de sistemas legados é raramente uma possibilidade cogitada devido aos altos investimentos realizados para sua implantação, sua modificação e integração com outros sistemas é a única alternativa viável. Além disso, os sistemas legados não devem ser encarados como *softwares* ultrapassados, pois apesar de seu envelhecimento, englobam um conhecimento sofisticado do negócio que incorpora vários outros aspectos técnicos e sociais. A manutenção destes sistemas evita a reconstrução de soluções já conhecidas e também os riscos inerentes à sua substituição, mas também é mais complexa à medida em que se tornam mais velhos (SOMMERVILLE, 2001: 583).

Temos portanto que um dos grandes desafios da área de TI é a integração de sistemas, tarefa que deve equacionar o equilíbrio entre o desenvolvimento de novas aplicações, a manutenção dos sistemas legados e a interoperabilidade adequada destes sistemas. Neste sentido, tem sido cada vez mais freqüente a utilização de *middlewares*, ou seja, *softwares* de propósito geral, compostos geralmente de um conjunto de padrões e bibliotecas de classes prontas, que suportam a comunicação entre componentes e a troca de informações (SOMMERVILLE, 2001: 314). Um *middleware* proporciona uma infra-estrutura de comunicação padronizada, onde diferentes sistemas, utilizando modelos de dados distintos, podem ser integrados de maneira relativamente transparente, bastando que sejam "adaptados" para comunicar-se com a infra-estrutura escolhida, o que simplifica as modificações e facilita futuras integrações. Exemplos populares de *middleware* são as implementações comerciais dos padrões abertos CORBA (OMG, 2004a) e JMS (SUN, 2004a), ou o uso de soluções proprietárias como Microsoft .NET (MS, 2004a) ou IBM MQSeries (IBM, 2004a).

Segundo Soley, a adoção de um *middleware* em especial não garante uma solução definitiva, pois as alterações ininterruptas nas necessidades das empresas farão surgir situações nas quais conjuntos de aplicações operando sobre diferentes *middlewares* devam ser integradas. Além disso, a evolução contínua das tecnologias também fará emergirem novos padrões e produtos mais adequados do que as soluções existentes, impondo outra vez a necessidade de integrarem-se sistemas construídos de forma distinta. Com isto percebemos que as questões relacionadas com a integração e interoperabilidade de sistemas têm suas raízes na diversidade de plataformas existentes e que tais problemas nunca poderão ser resolvidos através de uma solução única e definitiva (SOLEY et al., 2000).

Dentro deste cenário, existe uma grande preocupação quanto à preservação dos investimentos feitos no desenvolvimento do *software*, ou seja, a utilização de estratégias que garantam a integração dos sistemas existentes com aqueles que serão construídos, que sejam flexíveis quanto a mudanças necessárias na infra-estrutura utilizada e que estendam o ciclo de vida das aplicações construídas, reduzindo retrabalhos, custos de manutenção e aumentando o retorno dos investimentos realizados (FRANK, 2004; SOLEY et al., 2000).

APRESENTANDO A MDA

Projetistas experientes de aplicações usualmente investem mais tempo na construção de modelos do que em atividades de programação, pois a definição e o uso de modelos precisos e completos facilitam o desenvolvimento de sistemas corporativos dentro dos prazos e custos previstos, mesmo quando tais sistemas são grandes e complexos (KLEPPE et al., 2003). Considerando a demanda contínua das mudanças necessárias nas aplicações corporativas e a rapidez da evolução da tecnologia, temos que o cenário de desenvolvimento de soluções de TI corporativas é bastante complicado e dinâmico, dificultando a definição de modelos abrangentes e duradouros para os sistemas necessários.

Dentro deste panorama, a OMG (*Object Management Group*) propôs a MDA (*Model Driven Architecture*), uma estratégia para a construção de sistemas de TI onde são separadas a especificação das funcionalidades do sistema da especificação da implementação destas funcionalidades em uma plataforma de tecnologia particular (MILLER; MUKERJI, 2001: 3). Deseja-se que a MDA seja capaz de: especificar um sistema de modo independente da plataforma a ser adotada; especificar características das várias plataformas; permitir a escolha de uma plataforma particular; e permitir a transformação da especificação deste sistema numa implementação para a plataforma escolhida (MILLER et al., 2003).

A MDA representa uma visão das necessidades de interoperabilidade expandida para abranger completamente o ciclo de desenvolvimento de aplicações (MILLER; MUKERJI, 2001: 3), e permite que os desenvolvedores construam sistemas de acordo com a lógica de negócios e os dados existentes, independentemente de qualquer plataforma particular (KLEPPE et al., 2003), cujos detalhes tecnológicos são considerados irrelevantes na definição dos aspectos essenciais da funcionalidade desejada (MILLER; MUKERJI, 2001: 5).

Como as novas tecnologias vêm oferecer benefícios tangíveis para as companhias, muitas destas não podem “se dar ao luxo” de não empregar as inovações disponíveis (KLEPPE et al., 2003), assim a MDA foi concebida para auxiliar tais organizações na adoção rápida de novos conceitos e soluções tecnológicas, sem necessitar reconstruir inteiramente seus sistemas, pois é neutra em termos de linguagem, plataforma e fornecedor (FRANK, 2004; SOLEY et al., 2000). Novos sistemas podem então ser construídos com *middlewares* mais atuais, sem comprometer sua interoperabilidade com sistemas existentes (SIEGEL et al., 2001).

A MDA é um *framework* para desenvolvimento de *software* centrado na definição de modelos formais, cuja chave de compreensão e utilização é a importância dos modelos durante o processo de desenvolvimento de *software*. Dentro da arquitetura proposta pela MDA o processo de desenvolvimento de *software* é dirigido completamente pelas atividades de modelagem do sistema em diferentes níveis de abstração (KLEPPE et al., 2003; MELLOR et al., 2004).

O *framework* MDA é baseado grandemente na UML (*Unified Modeling Language*)

e também em outros padrões adotados pela indústria de *software*, tais como a MOF (*Meta-Object Facility*), CWM (*Common Warehouse Meta-model*) e XMI (XML *Metadata Interchange*). A UML é uma linguagem de modelagem de sistemas, cujo padrão é composto por uma linguagem gráfica e textual bem definida, adotado de fato pela indústria. O papel da UML é especificar a estrutura, funcionalidade e comportamento dos sistemas. A MOF também tem papel central na MDA, pois **unifica a notação da UML** (BOOCH, 2004; THOMAS, 2004), além de especificar a gerência dos modelos nos repositórios. A CWM padroniza a representação de modelos de bancos de dados (*schemas*) e suas transformações, bem como modelos para OLAP (*On-Line Analithical Processing*) ou mineração de dados (*data mining*). Já a XMI corresponde a um formato de troca para modelos baseado na MOF e também no XML. Tais padrões permitem a visualização, o armazenamento e a troca de projetos de *software* e modelos (KLEPPE et al., 2003; SIEGEL et al., 2001; SOLEY et al., 2000).

Mas diferentemente dos modelos puramente UML, a proposta da MDA é promover a criação de modelos abstratos que possam ser processados automaticamente (*machine-readable models*), desenvolvidos independentemente das tecnologias de implementação e também armazenados em repositórios padronizados. Ferramentas apropriadas (MDA *tools*) poderiam acessar tais modelos, transformando-os automaticamente em esquemas, esqueletos de código, código integrável, *scripts* de implantação, entre outros elementos (KLEPPE et al., 2003).

Segundo Bézenin (2001), um dos caminhos possíveis de evolução tecnológica a partir da orientação a objetos é o que poderia ser chamado "engenharia de modelagem", que dá aos modelos o *status* máximo, tal como anteriormente feito para as classes e objetos. A mudança essencial reside no fato de tais modelos deixarem de ser apenas peças de documentação, passando a ser usados diretamente para conduzir uma nova geração de ferramentas. A "engenharia de modelagem" trata o desenvolvimento de *software* como um conjunto de transformações sobre uma sucessão de modelos, partindo do levantamento de requisitos até a implementação e distribuição (THOMAS, 2004). O argumento de que estas tarefas poderiam ser completamente automatizadas é sustentado por muitos daqueles que contribuíram com a definição da MDA (MILLER et al., 2003).

Conceitualmente a MDA unifica e simplifica a quase totalidade das etapas do processo de desenvolvimento de *software*, principalmente aquelas relacionadas a modelagem, projeto, implementação e integração de aplicações, pois a definição do *software* se dá como um modelo funcional e comportamental separado dos elementos tecnológicos que serão posteriormente utilizados (OMG, 2004b; SIEGEL, 2002).

DESENVOLVIMENTO COM MDA

O ciclo de desenvolvimento de sistemas através da MDA é relativamente semelhante a outros processos de desenvolvimento de *software*, ou seja, compreende fases de levantamento de requisitos, análise, projeto, codificação, teste e implantação. Uma das maiores diferenças reside na natureza dos artefatos criados durante o processo de desenvolvimento, que segundo a MDA devem ser modelos formais que possam ser processados automaticamente (KLEPPE et al., 2003). Existem três modelos ou visões (*viewpoints*) na MDA:

- Modelo computacional independente ou *Computacional Independent Model* (CIM);
- Modelo independente de plataforma ou *Platform Independent Model* (PIM); e
- Modelo específico de plataforma ou *Platform Specific Model* (PSM).

O código (*code*), embora não constitua um modelo propriamente dito, também é um elemento importante nesta arquitetura, pois representa seu resultado concreto, como veremos adiante.

MODELO COMPUTACIONAL INDEPENDENTE (CIM)

O CIM representa uma visão do sistema de um ponto de vista computacional independente, que descreve apenas os requisitos gerais de funcionamento do sistema, isto é, não exibe detalhes da estrutura do sistema, mas considera um determinado tipo de sistema cujo ambiente possui necessariamente características comuns a outros sistemas do mesmo domínio computacional. Por isso também é denominado modelo de domínio (*domain model*) ou modelo de núcleo (*core models*) (MILLER et al., 2003; SOLEY et al., 2000).

Sistemas em tempo real possuem requisitos bastante distintos de sistemas transacionais ou sistemas distribuídos baseados em componentes, assim a primeira visão do sistema deverá considerar em qual destes domínios se dará sua operação. Embora esta visão contenha informações importantes, não é tratada usualmente como um elemento essencial, diferente do PIM e PSM, que são fundamentais na MDA.

MODELO INDEPENDENTE DE PLATAFORMA (PIM)

O primeiro artefato especificado pela MDA é o PIM (*Platform Independent Model*) ou modelo independente de plataforma, cujo alto grau de abstração deve permitir representar o sistema de modo independente de qualquer tecnologia de implementação. O PIM é um modelo declarativo formal da estrutura funcional do sistema (MILLER; MUKERJI, 2001: 6), ou seja, tem foco na operação do sistema e,

por isso, deve oferecer a melhor descrição possível para suportar o negócio em questão, mas sem considerar qual plataforma será utilizada ou como tal sistema será construído (KLEPPE et al., 2003; SIEGEL et al., 2001), ou seja, é neutro tecnologicamente. É um modelo destinado a preservar a informação essencial a respeito do projeto da aplicação, sua arquitetura e infra-estrutura (BOOCH, 2004).

O PIM também constitui um modelo bastante rico em termos semânticos, pois pode ser representado gráfica ou textualmente em termos da UML e suas extensões (tais como a OCL - *Object Constraint Language* - que facilita a indicação de restrições), dando ao projetista meios de expressar mais precisamente suas intenções e, com isso, reduzindo o trabalho das etapas posteriores. Outro aspecto importante é que o PIM poderá ser armazenado em um repositório através da MOF, possibilitando sua recuperação e processamento posteriores.

MODELO ESPECÍFICO DE PLATAFORMA (PSM)

O segundo artefato determinado pela MDA é o PSM (*Platform Specific Model*) ou modelo específico de plataforma. Este modelo, também expresso através da UML, deve representar o sistema em termos de construções apropriadas de uma tecnologia particular, ou seja, considerando a operação descrita pelo PIM e também detalhes específicos da implementação em termos da tecnologia selecionada (KLEPPE et al., 2003; MILLER; MUKERJI, 2001; MILLER et al., 2003; SIEGEL, 2002). Assim um PSM voltado para a tecnologia EJB descreverá o sistema utilizando suas estruturas, tais como *home interface*, *session bean* ou *entity bean*, enquanto um PSM voltado para *Web Services* incluirá termos como SOAP, *provider* ou XML *schemas*. Um PSM será, como um PIM, armazenado em repositório através da MOF.

Cada PSM deve ser o resultado de uma transformação automática do PIM em termos de uma tecnologia específica (*standard mapping*), implicando que um mesmo PIM pode originar diferentes PSM, conforme as transformações aplicadas (KLEPPE et al., 2003; SIEGEL et al., 2001; SOLEY et al., 2000). Isto significa que uma determinada plataforma deverá ser escolhida, implicando na seleção de mecanismo de mapeamento particular, para a transformação de um artefato PIM em um PSM. Conforme Miller e Mukerji (2001), a transformação de um PIM em um PSM CORBA exigiria a escolha de elementos específicos definidos num UML *profile*, no qual as classes UML representariam as interfaces, tipos e outras construções do CORBA através de estereótipos (*stereotypes*).

A operação de transformação de um PIM em um PSM é o passo crucial do processo de desenvolvimento MDA (KLEPPE et al., 2003), pois representa o maior ganho oferecido, dado que é relativamente comum que um mesmo sistema tenha que operar em diferentes plataformas, evitando a repetição dos esforços de desenvolvimento (SIEGEL et al., 2001).

CÓDIGO

O código é o produto final da MDA e deve ser o resultado da transformação de um dado PSM considerando uma tecnologia de implementação específica. A geração de código é uma etapa relativamente simples dada a proximidade do PSM com a tecnologia particular em uso (KLEPPE et al., 2003). Em algumas circunstâncias, quando existir suporte para múltiplas linguagens de programação, deverá ser efetuada a seleção da linguagem de implementação desejada. Além do código também poderão ser gerados outros artefatos necessários ao sistema, tais como arquivos de configuração, entradas de registro, *scripts*, etc.

Após a geração do código será provável a necessidade de alguns ajustes e complementações, que deverão ser feitos por uma equipe de programação, mas espera-se que em quantidades bastante menores do que com as atuais ferramentas.

TRANSFORMAÇÕES AUTOMÁTICAS

O grande diferencial da MDA reside na forma de realização das transformações entre os modelos PIM, PSM e o código. Tradicionalmente as transformações entre modelos de um processo de *software* são realizadas manualmente. Diferentemente na MDA, os modelos deverão ser usados para geração automática da maior parte do sistema (SIEGEL et al., 2001). Na MDA todas transformações devem ser realizadas automaticamente por ferramentas apropriadas, o que pode significar maior rapidez e flexibilidade na geração de aplicações de melhor qualidade, caracterizando assim os benefícios imediatos de sua aplicação (KLEPPE et al., 2003). As transformações entre modelos ou mapeamento (*mappings*) são entendidas como o conjunto de regras e técnicas aplicadas em um modelo de modo que seja obtido um outro com as características desejadas. A MDA considera a existência de quatro tipos de transformações diferentes (MILLER; MUKERJI, 2001):

- PIM para PIM. Utilizada para o aperfeiçoamento ou simplificação dos modelos sem a necessidade de levar em conta aspectos dependentes de plataforma.
- PIM para PSM. Transformação “padrão” do modelo independente de plataforma para outro específico durante o ciclo de desenvolvimento típico de aplicações.
- PSM para PSM. Esta transformação permite a migração da solução entre plataformas diferentes, bem como o direcionamento de partes da implementação para tecnologias específicas, usadas por questões de interoperabilidade ou benefícios obtidos através do uso de certas plataformas.
- PSM para PIM. Quando é necessário obter-se uma representação neutra em termos de tecnologia de soluções específicas.

Entre as transformações pode existir o processo de marcação (*marking*), que constitui uma forma "leve" e pouco intrusiva de extensões dos modelos com elementos voltados a facilitar uma transformação particular (MELLOR et al., 2004). Por exemplo, um PIM (sem marcações) pode receber anotações destinadas a uma plataforma A ou B, originando *marked* PIMs e mantendo o PIM original sem qualquer "contaminação".

Embora hoje existam muitas ferramentas capazes de gerar algum código a partir de modelos particulares, usualmente tal código é pouco mais que um "esqueleto" (um *template*), exigindo que a finalização da implementação seja feita através de atividades de programação convencional. Kleppe et al. (2003) afirmam que não dispomos atualmente de ferramentas que sejam capazes de realizar completamente a transformação PIM para PSM e deste para código, requisitando a intervenção manual de programadores para finalização dos modelos e da codificação, mas que as ferramentas MDA existentes são capazes de gerar protótipos funcionais, embora simplificados, do sistema acelerando o ciclo de desenvolvimento. Miller e Mukerji (2001) indicam que as transformações completas são possíveis em ambientes dotados de restrições, dentre as quais: ausência de legados a considerar; o modelo de partida é semanticamente rico; e os algoritmos de transformação são de alta qualidade.

A geração de código não é considerada o aspecto mais importante da MDA (KLEPPE et al., 2003; MILLER; MUKERJI, 2001), pois este é bastante próximo à estrutura declarativa e atributos, sendo simples a estrutura funcional de métodos de acesso (*setter and getter methods*); por outro lado é bastante complexa a geração das características comportamentais do sistema como um todo. Exatamente por isso, projetistas que utilizam a MDA não devem esperar que a primeira versão gerada para o sistema seja perfeita, pois o próprio processo MDA assume a necessidade de múltiplas iterações entre o projeto e a implementação obtida, ou seja, a necessidade de refinamento como meio de produzirem-se sistemas de qualidade (MILLER; MUKERJI, 2001; SIEGEL, 2002).

BENEFÍCIOS DA UTILIZAÇÃO DA MDA

Através do emprego da MDA espera-se obter os seguintes benefícios:

· **Produtividade**. Na MDA o foco da atividade de desenvolvimento é deslocado para a construção do PIM, pois os PSM deverão ser obtidos através de transformações automáticas suportadas pelas ferramentas MDA. Embora as transformações envolvidas sejam complexas, requerendo especialistas em sua especificação, depois de definidas podem ser aplicadas em diferentes contextos, possibilitando o retorno do investimento em termos de produtividade. Eventualmente algumas transformações comuns na indústria podem ser adquiridas ou mesmo colocadas em domínio público. Outro ganho possível se relaciona ao fato de a construção do PIM

não envolver detalhes específicos de implementação, concentrando a atividade na correta definição do sistema. Além disso, a quantidade de código que deverá ser manualmente gerada é reduzida (KLEPPE et al., 2003: 9).

· **Portabilidade**. Como um mesmo PIM pode ser transformado em diferentes PSMs, possibilitando que um sistema possa operar em diferentes plataformas, então a construção do PIM é portável para qualquer plataforma para a qual exista a definição da transformação PIM para PSM especificamente envolvida (KLEPPE et al., 2003: 10). Para plataformas menos populares é possível a definição particular das transformações desejadas, assim como o surgimento de novas plataformas irá requerer apenas a definição do mapeamento específico, o que em tese maximiza a portabilidade de qualquer PIM com relação às tecnologias atuais e futuras (BOOCH, 2004; SOLEY et al., 2000).

· **Interoperabilidade**. Os PSMs dirigidos para plataformas diferentes não são interoperáveis diretamente. Mas para não deixar esta questão em aberto, a MDA também prevê a existência de *bridges* entre os PSMs e também entre o código de cada plataforma. A própria geração destas *bridges* pode ser automatizada pois é conhecida a origem de cada elemento do PSM em termos de sua definição no PIM e, por conseguinte, qual o elemento correspondente no outro PSM de interesse; sistemas legados podem ser integrados através DE adaptadores (*wrappers*) especificados em termos da MDA (SOLEY et al., 2000). Um PIM baseado em uma máquina virtual não exigirá transformações, mas sim o PIM da máquina virtual, o qual deverá ser mapeado em um PSM destinado a uma plataforma específica, mantendo o sistema independente de plataforma e interoperável naquelas onde exista uma máquina virtual apropriada (MILLER et al., 2003).

· **Manutenabilidade**. Através de alterações no PIM do sistema é possível a geração de novos PSMs e código correspondente muito rapidamente, agilizando e barateando os procedimentos de manutenção do sistema. Com isto correções, adaptações ou mesmo a adição de novas funcionalidades tornam-se tarefas mais simples de serem realizadas, prolongando a vida útil do sistema (SOLEY et al., 2000). Espera-se que boas ferramentas MDA permitam manter a correspondência adequada entre PIM e PSM nas situações em que um ou outro destes modelos sejam modificados.

· **Teste e Validação**. Da mesma forma que os modelos construídos podem ser automaticamente transformados em outros modelos e também em código, é possível que sejam validados segundo critérios pré-definidos e também testados dentro dos parâmetros das plataformas em que deverão operar (MILLER et al., 2003). Isto também abre algumas possibilidades em termos de simulação, reforçando o grande potencial da MDA na geração de sistemas mais robustos, portáteis e adequados às necessidades identificadas.

· **Documentação**. A MDA é baseada na construção de modelos formais, que sob muitos aspectos correspondem a uma importante documentação do sistema. O PIM é o artefato mais importante, pois corresponde a uma documentação de alto

nível (BOOCH, 2004). Além disso, como tais modelos podem ser visualizados, armazenados e processados automaticamente, não são abandonados após a finalização do sistema, pois tanto o PIM, no nível de abstração mais alto, quanto o PSM, num nível de abstração intermediário, podem ser reutilizados para a incorporação de alterações no sistema ou mesmo sua migração para outras plataformas. Embora a formalização dos modelos necessários a MDA cumpra o papel de documentação do sistema, outras informações deverão ser adicionadas aos modelos, tais como os problemas e necessidades diagnosticadas, bem como o racional das escolhas efetuadas (KLEPPE et al., 2003: 11).

Tais benefícios se refletem diretamente na redução dos custos de desenvolvimento, redução do tempo de desenvolvimento, aumento da qualidade das aplicações produzidas, aumento do retorno dos investimentos realizados e aceleração do processo de adoção de novas tecnologias, bem como simplificação dos problemas associados com a integração de sistemas (OMG, 2004b).

Miller e Mukerji também enfatizam o fato de os modelos da MDA serem exibidos num maior ou menor nível de detalhes, permitindo a análise, conversão e comparação de modelos. Além disso, o mapeamento explícito entre os modelos, que possibilita sua transformação automática, permite a rastreabilidade e controle de versão dos artefatos utilizados (MILLER & MUKERJI, 2001: 5-8).

A aplicação de padrões de projeto também é um dos benefícios a serem obtidos com a MDA, pois na transformação PIM para PSM é possível que padrões conhecidos possam ser aplicados no sistema em questão (MILLER et al., 2003), automatizando algumas das etapas de construção dos componentes do sistema na tecnologia escolhida (ALEXANDRE, 2003).

Booch (2004) defende o emprego da MDA afirmando que seus praticantes não necessitam ser profissionais altamente gabaritados em UML, cujo trabalho iterativo em equipe deverá resolver as questões mais sérias das abstrações envolvidas na elaboração do PIM. Também argumenta que o uso de padrões de projeto é favorecido, que os ganhos em portabilidade e interoperabilidade são enormes e que com o desenvolvimento das ferramentas MDA os ganhos serão ainda maiores.

Outros aspectos, menos tangíveis, são: que a modelagem em alto nível favorece as tarefas de validação, pois os detalhes de implementação não estão inclusos no PIM; que a aplicação ou integração de plataformas diferentes tornam-se mais claramente definidas, facilitando a produção de implementações particulares (MILLER; MUKERJI, 2001: 8).

DIFICULDADES NA APLICAÇÃO DA MDA

A UML é uma linguagem de modelagem, que originalmente se destinava a oferecer uma forma visual de comunicação para representação dos principais conceitos e elementos de um sistema. Embora seja utilizada amplamente pela indús-

tria de desenvolvimento de *software*, que reconhece suas muitas qualidades, deixa algumas lacunas: não possui uma semântica completamente formalizada, o que deixa brechas para diferentes interpretações e implementações de suas representações; não existe uma implementação de referência cuja semântica permita garantir a correta interpretação e transformação de seus modelos; não possui um formato de intercâmbio definido, o que dificulta o compartilhamento de seus modelos entre as ferramentas que a suportam; e requer o uso de várias outras linguagens de extensão para que sejam expressos elementos que não fazem parte do seu escopo (as restrições de atributos, por exemplo) ou para que seja armazenada e processada (Fowler, 2004; Thomas, 2003; Thomas, 2004).

Para Thomas (2004), apesar dos argumentos da "engenharia de modelagem", um meta-modelo, tal como o PIM, permite a adição de novas funcionalidades com relativa facilidade, mas por si só não garante que tais funcionalidades se articulem de modo coerente. Assim continua nas mãos dos projetistas avaliar e garantir tal consistência, o que continua a exigir a alocação de profissionais experientes, e assim percebemos que a MDA não contribui de forma efetiva neste sentido. Cook (2004) argumenta que a neutralidade do PIM em termos de plataforma poderá significar, em última instância, perdas em termos de performance, o que poderá comprometer o desenvolvimento de sistemas com alto volume de transações ou em tempo real. Conforme Medvidovic et al., apenas a UML não é suficiente para "capturar" todos os aspectos necessários à modelagem de uma arquitetura de *software* completa, necessitando de extensões ADL (*Architecture Description Languages*). O direcionamento da modelagem para domínios específicos (*domain specific modeling*) poderia trazer ganhos para MDA (Agrawal, 2003; Cook, 2004; Mellor et al., 2004), possibilitando a geração de aplicações mais bem ajustadas para suas efetivas condições de utilização.

Existem ainda as dificuldades na transformação PIM para PSM, pois além da oferta de uma enorme diversidade de plataformas (CCM, J2EE, .NET, etc.), se tal transformação não for precisa o suficiente, a geração de código não será possível ou não produzirá os ganhos de produtividade esperados. Cada uma destas plataformas possui uma API bastante ampla, complexa e nem sempre bem documentada; é muito difícil a construção dos *standard mapping* para realização das transformações automáticas. Efetivamente só foram demonstrados os mapeamentos para CORBA (OMG, 2004c) e J2EE (Alexandre, 2003; Cook, 2004). Existem sérias dúvidas quanto a capacidade e interesse da indústria de TI na confecção de UML *profiles* para todos os domínios específicos (Thomas, 2003). Ainda nesta questão, Fowler (2004) não vê ganhos adicionais da MDA quando comparada com os resultados do desenvolvimento sustentado por padrões, bibliotecas e *frameworks*.

Observamos também que dentre as diversas ferramentas oferecidas pelos inúmeros fornecedores que "adotaram" a MDA, nenhuma é capaz de realizar completamente as transformações entre PIM e PSM, ou mesmo efetuar a geração de código a partir do PSM, pelo menos não como idealizado pela proposta desta

arquitetura. Efetivamente as ferramentas existentes apenas adaptam os métodos de trabalho previamente existentes numa "roupagem" MDA. Este cenário é desconfortável, pois remete a uma situação semelhante durante a década de 80, quando muito foi "prometido" para as ferramentas CASE (*Computer Aided Software Engineering*) e efetivamente pouco foi implementado e disponibilizado, mesmo que comercialmente (COOK, 2004; FOWLER, 2004; THOMAS, 2004).

Percebemos, finalmente, que nos mesmos argumentos que incentivam o uso da MDA residem algumas dúvidas, bastante sérias, quanto aos ganhos que podem ser obtidos.

CONCLUSÃO

A proposta da MDA é oferecer meios concretos para melhorar a produtividade, portabilidade, interoperabilidade, manutenabilidade e documentação de sistemas. Para isto esta arquitetura estabelece seu foco na modelagem da funcionalidade e comportamento de aplicações e sistemas distribuídos, separando os aspectos particulares das tecnologias que serão de fato usadas em sua implementação. Desta maneira é necessária a criação de modelos detalhados e representativos dos aspectos funcional-comportamental e tecnológico, nos quais se posicionam a UML e as demais linguagens que sustentam a MDA.

A UML, elemento central para a construção de modelos MDA, provê uma grande variedade de elementos para o projetista de *software*, tais como múltiplas visões inter-relacionadas do sistema, uma semântica que permite a expressão de metamodelos e uma série de linguagens de extensão que permitem suprir outras necessidades. Mas como discutido, além de sua complexidade, a UML ainda não permite a definição semanticamente completa de modelos para qualquer domínio.

Embora seja coerente a proposta da OMG na utilização de outras linguagens complementares para expressão, armazenamento e processamento dos modelos necessários à operacionalização da MDA, isto adiciona uma complexidade indesejável ao processo.

Outro benefício proposto é a automação da tarefas de transformação através do emprego de técnicas de mapeamento dos modelos, mas neste ponto também residem dúvidas sobre a real capacidade das ferramentas oferecidas hoje e nos próximos anos, o que compromete parcialmente os ganhos prometidos.

Considerando este contraponto, concluímos que o principal benefício da MDA é, efetivamente, os ganhos de qualidade e neutralidade proporcionados pelas atividades de modelagem, especialmente quando realizada independente da plataforma. A automatização do processo de transformação destes modelos em outros tecnologicamente particularizados poderá trazer grandes ganhos, justificando os trabalhos de pesquisa realizados nesse sentido. Até lá, vale a pena utilizar os conceitos da "engenharia de modelagem", mesmo que seus procedimentos sejam efetuados manualmente.

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, A. *Metamodel based model transformation language*. In: *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, p. 386–387. ACM Press, 2003.

ALEXANDRE, T. *Using design patterns to build dynamically extensible collaborative virtual environments*. In: *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, p. 21–23. Computer Science Press, Inc., 2003.

BOOCH, G. *MDA: A motivated manifesto?* In: Software Development Magazine. On-line: http://www.sdmagazine.com/documents/s=9224/sdm0408a/sdm0408a.html, recuperado em 22/10/2004.

BÉZIVIN, J. *From Object Composition to Model Transformation with the MDA*. In: *Proceedings of TOOLS'USA*. IEEE Press, 2001.

COOK, S. *Domain-Specific Modeling and Model Driven Architecture*. On-line: http://www.bptrends.com/publicationfiles/01-04\%20COL\%20Dom\%20Spec\%20Modeling\%20Frankel-Cook.pdf, recuperado em 02/11/2004. MDA Journal, 2004.

FOWLER, M. *Model Driven Architecture*. On-line: http://martinfowler.com/bliki/ModelDrivenArchitecture.html, recuperado em 02/11/2004.

FRANK, K. *Keeping your business relevant with Model Driven Architecture (MDA)*. Borland White Paper. On-line: http://www.omg.org/borland-mda-wp, recuperado em 22/10/2004.

HART, J. *Connecting your applications without complex programming*. On-line: http://www.ibm.com/software/integration/wmq/, recuperado em 30/10/2004., Sept. 2003. IBM White Paper.

IBM. IBM Message Queue Series. On-line: http://www.ibm.com/software/integration/wmq/, recuperado em 30/10/2004.

KLEPPE, A.; WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture - Practice and Promise*. Addison-Wesley, Reading, MA, 2003.

MEDVIDOVIC, N.; RESENBLUM, D. S.; REDMILES, D. F.; ROBBINS, J. E. *Modeling software architectures in the unified modeling language*. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(1): 2–57, 2002.

MELLOR, S.; SCOTT, K.; UHL, A.; WEISE, D. *MDA Distilled*. Addison-Wesley, Reading, MA, 2004.

MICROSOFT. *Microsoft .NET Platform*. On-line: http://www.microsoft.com/net/, recuperado em 28/09/2004.

MILLER, J.; MUKERJI, J. *Model Driven Architecture (MDA)*. Objetc Management Group Architecture Board ORMSC. On-line: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, recuperado em 26/10/2004, 2001.

_____. (Ed.). MDA Guide Version 1.0.1. Objetc Management Group. On-line: http://www.omg.org/docs/omg/03-06-01.pdf, recuperado em 22/10/2004.

OMG. *Common Object Request Broker Architecture*. On-line: http://www.omg.org/technology/documents/formal/corba_iiop.htm, recuperado em 26/10/2004. Object Management Group White Paper, 2004a.

OMG. *MDA (Model-Driven Architecture) Executive Overview*. On-line: http://www.omg.org/mda/executive_overview.htm, recuperado em 22/10/2004. Object Management Group White Paper, 2004b.

OMG. *UML Profile for CORBA*. On-line: http://www.omg.org/technology/documents/formal/profile_corba.htm, recuperado em 22/10/2004. Object Management Group White Paper, 2004c.

SIEGEL, J. & OMG Staff Strategy Group. *Developing in OMG's Model-Driven Architecture*. On-line: ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf, recuperado em 26/10/2004. Object Management Group White Paper, 2001.

SIEGEL, J.. *Using OMG's Model-Driven Architecture (MDA) to integrate Web Services*. On-line: http://www.omg.org/mda/, recuperado em 26/10/2004. Object Management Group White Paper, 2002.

SOLEY, R. & OMG Staff Strategy Group. *Model Driven Architecture*. On-line: http://www.omg.org/cgibin/doc?omg/00-11-05, recuperado em 26/10/2004. Object Management Group White Paper, 2000.

SOMMERVILLE, I. *Software Engineering*. Addison-Wesley, Harlow, England, 6th edition, 2001.

SUN. *Sun Java Message System*. On-line: http://www.java.sun.com/products/jms/, recuperado em 28/09/2004.

THOMAS, D. *Unified or universal modeling language?* In: *Journal of Object Technology*, 2(1): 7–12, Jan.-Feb. 2003.

THOMAS, D. *MDA: Revenge of the modelers or UML utopia?* In: IEEE Software, p. 22–24, May-June 2004.