2
Fundamentos

Neste capítulo serão abordados de forma introdutória os principais conceitos relacionados com replicação de dados. Em seguida, será abordada a metodologia de projeto orientado a domínio que garante maior qualidade conceitual à arquitetura. Por fim, será abordado o paradigma de desenvolvimento de software orientado a agentes, importante no entendimento conceitual da arquitetura.

2.1. Replicação de Dados

Replicação é uma estratégia para tornar serviços tolerantes a falhas. Esta estratégia consiste em manter componentes redundantes de maneira que uma falha dispara um processo de troca de componentes com o objetivo de tornar o serviço disponível durante o maior tempo possível. Em sistemas de informação, estes componentes de serviço envolvem aplicações, dados, e toda infra-estrutura de hardware, incluindo a infra-estrutura de rede. Em particular, a replicação de dados aumenta a disponibilidade e pode aumentar a performance do acesso aos dados.

A replicação de dados pode ser *ativa* ou *passiva*. Na replicação *ativa*, a mesma operação é executada em cada cópia dos dados. Já na replicação *passiva* os dados resultantes de cada operação são replicados. Assim, os dados são divididos em *cópias mestres* e *cópias escravas*. As aplicações operam em cópias mestres. Já as cópias escravas são atualizadas via replicação e são utilizadas por aplicações apenas para leitura. Em caso de falha, uma cópia escrava poderá ser eleita para se tornar mestre.

O fluxo de replicação de uma cópia mestre para cópias escravas pode ser *síncrono* ou *assíncrono*. Num fluxo síncrono, todas as cópias são atualizadas numa única transação distribuída. Assim, as cópias permanecem equivalentes durante todo instante. Porém, há uma latência maior no processo de escrita porque cada transação só é fechada quando todas as cópias escravas terminam de aplicá-la. Além disso, uma transação aberta durante todo este intervalo irá bloquear recursos que poderiam ser utilizados concorrentemente por outras transações.

PUC-Rio - Certificação Digital Nº 0510977/CA

Já num fluxo assíncrono, as atualizações são enfileiradas e replicadas individualmente. Por um lado, cada transação possui um tempo menor de processamento. Por outro lado, pode haver atraso na equivalência entre as cópias.

O fluxo de replicação também pode ser classificado como unidirecional ou bidirecional. No fluxo unidirecional, as atualizações seguem apenas uma direção de uma base mestre para uma ou mais bases escravas. Já o fluxo bidirecional possibilita a existência de mais de uma base mestre. Dessa maneira, num fluxo bidirecional uma base pode receber e enviar atualizações.

Quanto à granularidade, a replicação pode ocorrer a cada operação nos dados (*on-line*), ou a cada intervalo temporal pré-definido (*off-line*). Na replicação *on-line* as cópias mantêm um pequeno ou nenhum atraso, enquanto a replicação *off-line* oferece maior desempenho para bases de dados com grande número de transações. Este trabalho propõe uma arquitetura de replicação passiva, assíncrona, unidirecional e *off-line.*

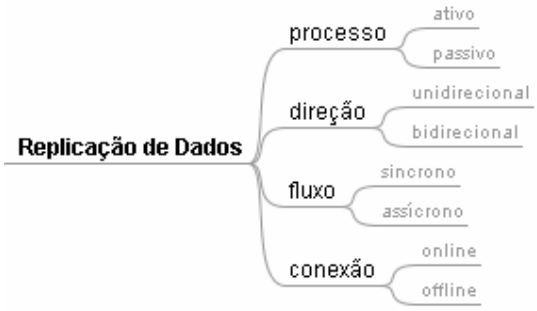

Figura 1. Estratégias de replicação de dados

2.2. Projeto Orientado a Domínio

Evans [EVANS, 2004] propõe que o desenvolvimento e manutenção de *software* complexo pode ser facilitado quando o domínio do software está isolado das tecnologias, ferramentas e ambientes envolvidos. Dessa forma, todas as pessoas que se relacionam com o software, incluindo analistas, desenvolvedores e usuários, podem obter ganhos advindos de uma melhor compreensão mútua.

Um primeiro passo para que isso se torne verdade é a definição de uma linguagem ubíqua para os envolvidos. Essa linguagem estará presente na interface

PUC-Rio - Certificação Digital Nº 0510977/CA

do software, no código, na documentação, nos manuais e deve estar consistente com as regras de negócio.

Para simplificar o desenvolvimento, é sugerido que o código que expressa o domínio do software esteja concentrado e isolado numa camada específica. Evans definiu um conjunto de padrões comuns em domínios. Os objetos do domínio foram diferenciados em *entities*, *value objects* e *aggregates*. *Entities* são objetos com uma identidade que permanece constante durante todos os estados do software. Já *value objects* são objetos imutáveis sem identidade. Sob esta classificação. Alunos e Carros são *entities*, enquanto endereços e cores são *value objects*. Já os *aggregates* agrupam *entities* e *value objects* logicamente de forma a aumentar a granularidade de objetos manipuláveis. Assim, diminue-se drasticamente as possibilidades de associações entre objetos, simplificando o modelo.

Para manipular o domínio, são utilizados *factories*, *repositories* e *services*. As *factories* e *repositories* são responsáveis pela criação e armazenamento de *aggregates*, respectivamente.

Já os *services* realizam algum comportamento que não está diretamente associado a nenhuma *entity* ou *value object*. Tipicamente, uma transação. Finalmente, para que um domínio se mantenha administrável, é recomendado modularizá-lo dividindo o domínio verticalmente.

PUC-Rio - Certificação Digital Nº 0510977/CA

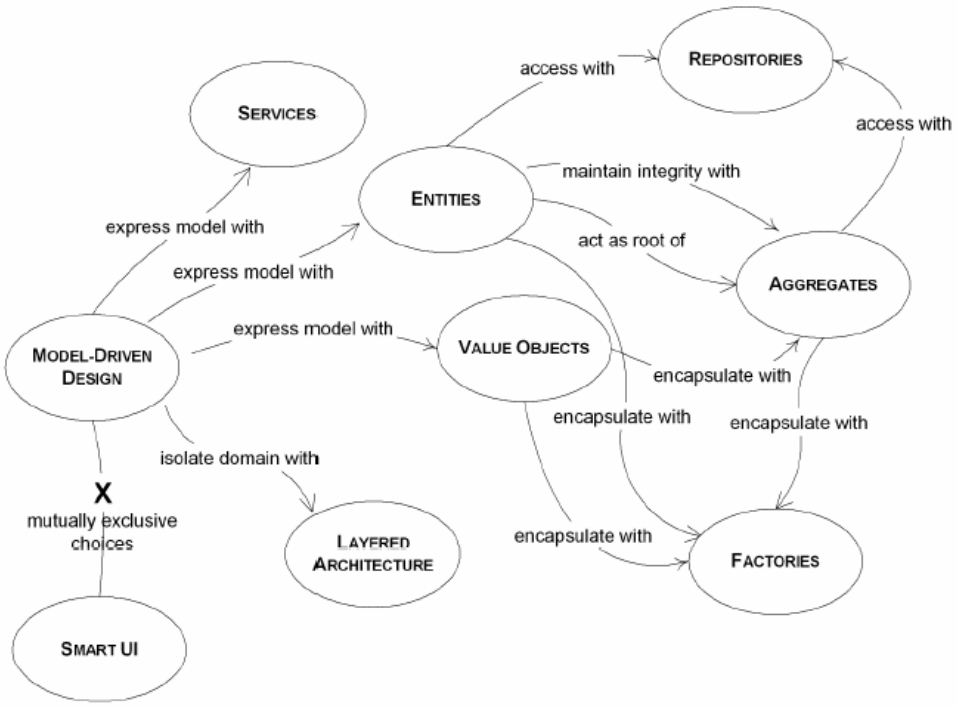

Figura 2. Projeto orientado a domínio

2.3. Software Orientado a Agentes

O desenvolvimento de *software orientado a agentes* está se estabelecendo como um importante paradigma para sistemas complexos onde se caracteriza uma grande diversidade de variáveis, decisões e ações possíveis. Tais sistemas são complexos para se pensar, ensinar e reproduzir comportamentos esperados em apenas uma linha de execução de instruções.

Em um sistema de comércio eletrônico, por exemplo, pode-se projetar agentes específicos para monitorar o estoque e solicitar pedidos automaticamente aos fornecedores, monitorar as vendas e criar promoções, dar atendimento personalizado aos clientes, ou mesmo gerenciar a segurança do negócio.

Assim, com agentes é possível concentrar as decisões de um módulo ou aspecto de um sistema a um responsável autônomo. Estes responsáveis são como "robôs virtuais" que agem em áreas específicas do problema. Para que isso se torne possível, os agentes devem estar munidos de pró-atividade, comunicação, aprendizado, negociação, coordenação, crenças, objetivos e outras características tipicamente humanas.

PUC-Rio - Certificação Digital Nº 0510977/CA

Um sistema multi-agente transforma um software de uma simples ferramenta à disposição dos humanos para se tornar parte ativa dos processos de negócio, atuando inclusive sem a interferência humana. A figura que segue ilustra a interação entre agentes, comunicando entre si e atuando em partes do ambiente.

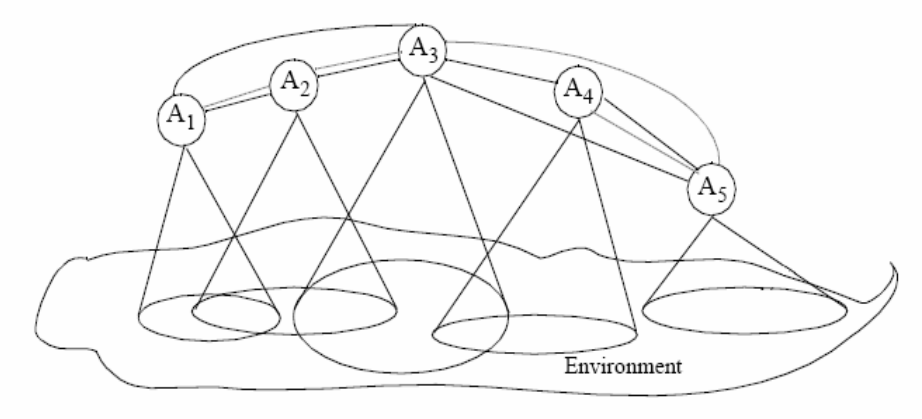

Figura 3. Atuação de agentes em um ambiente

PUC-Rio - Certificação Digital Nº 0510977/CA