# 3
# Arquitetura para Replicação de Bases Heterogêneas

Neste capítulo será abordada de forma conceitual a arquitetura proposta. Primeiramente, serão introduzidos os conceitos relacionados com bases heterogêneas. Em seguida, serão listados os principais requisitos para replicação de bases heterogêneas. A seguir, serão listadas as estratégias adotadas neste trabalho para atender estes requisitos. Logo após, serão abordados os agentes e utilitários que compõe a arquitetura e como eles cooperam no uso destas estratégias. Finalmente, serão detalhadas as camadas que compõe um serviço de replicação.



## 3.1. Base Heterogênea

Uma *base heterogênea* é caracterizada como sendo um conjunto de diferentes tipos de *módulos de dados*. Um módulo de dados pode ser uma partição de um banco de dados, um conjunto de documentos XML, planilhas, e-mails e outros [Anexo A].

Entre dois *módulos de dados* pode haver uma *relação de referência*. Um exemplo típico é um campo de uma tabela de um banco de dados que contém o endereço de um documento que, por sua vez, está armazenado em um determinado sistema de arquivos.

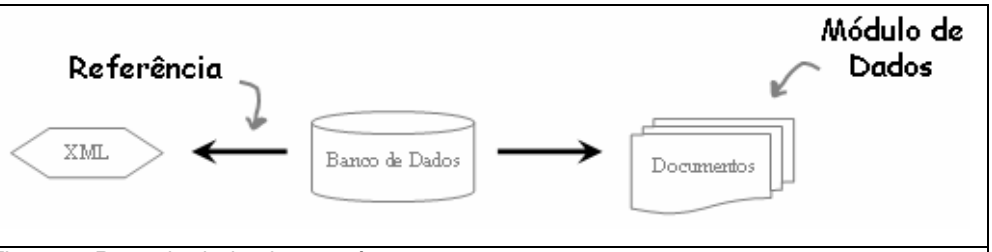


Figura 4. Base de dados heterogênea

Um serviço de replicação deve respeitar estas relações de referência de maneira a impedir que algum módulo entre em um estado inconsistente. Ao ser

replicada, uma base heterogênea pode agrupar módulos de dados vindos de diversas bases. Além disso, pode-se desejar replicar apenas alguns módulos de uma base.

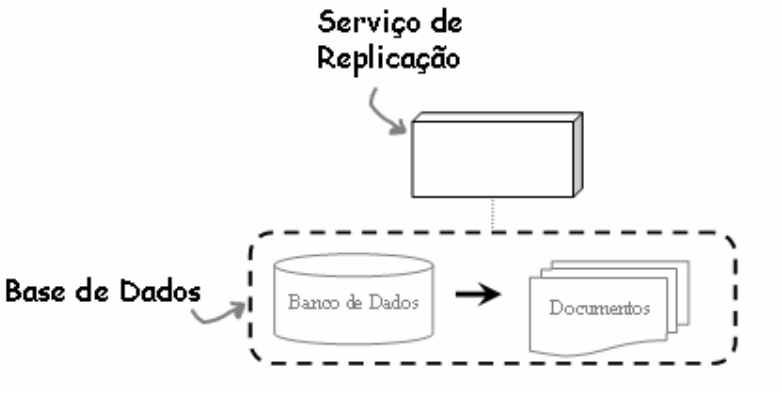


Figura 5. Base de dados e serviço de replicação

## 3.2. Serviço de Replicação de Bases Heterogêneas

Um serviço de replicação é um processo que atua em *background* repetindo as alterações de uma base de dados mestre em bases de dados escravas. Esta qualidade é intensificada com os seguintes requisitos:

- **Tolerância à falhas**. O serviço de replicação deve se manter ativo mesmo diante de falhas nas bases de dados. Estas falhas podem ser causadas por problemas como indisponibilidade no serviço de rede, por exemplo. Assim, após a rede ser restabelecida, o serviço de replicação deve ser capaz de continuar o processo sem intervenção humana.
- **Integridade transacional**. O serviço de replicação deve manter as propriedades *atomicidade, consistência, isolamento e durabilidade* (ACID) das transações ocorridas nas bases mestres. Assim, uma seqüência de operações de uma transação realizada numa base mestre deve ser aplicada transacionalmente na base escrava.
- **Integridade referencial.** Um serviço de replicação deve garantir que uma base heterogênea seja replicada mantendo a sincronia das atualizações de cada módulo de dados. Ou seja, um módulo de uma

base não pode estar mais atualizado que o outro módulo da mesma base. Formalmente, a cada instante, se um módulo de dados **M1** referencia um módulo de dados **M2**, então todos os dados de **M2** referenciados por **M1** devem estar atualizados juntamente com **M1**.

- **Continuidade**. O serviço de replicação deve permitir que o processo de replicação possa ser interrompido. Em caso de interrupção, as bases de dados devem se manter a integridade transacional e referencial. Quando o processo é restabelecido, a replicação deve iniciar no ponto interrompido, sem perda de dados.
- **Processamento independente**. A coordenação dos fluxos deve ser realizada de forma que a replicação de uma base não interfira na replicação de outra base. Ou seja, é esperado que a replicação de uma base de dados possa ser executada em paralelo com a replicação de outras bases sem interferências.

Este trabalho propõe um conjunto de soluções com o objetivo de satisfazer estes requisitos. O mapeamento entre os requisitos e suas respectivas soluções é ilustrado na Figura 6. As próximas seções detalharão estas soluções.

| | Tolerância a falhas | Integridade transacional | Integridade referencial | Continuidade | Processamento independente |
|---|---|---|---|---|---|
| Processamento em ciclos | X | | | X | |
| Versionamento | | X | X | | |
| Controle dos estados das bases | | X | | | X |
| Coordenação por multi-serviços | | | | | X |
| Processamento Assíncrono | | | | X | X |
| Operações transacionais | X | X | X | X | |

Figura 6. Mapeamento entre os requisitos e as soluções

### 3.2.1. Processamento em Ciclos

O processamento de cada base de dados pode ser realizado de forma cíclica. Ou seja, periodicamente uma base de dado mestre é consultada. Caso existam alterações deste a última consulta, estas alterações são lidas e enfileiradas para serem processadas.

Se a base de dados estiver indisponível, o processamento é interrompido. Porém, após um período de espera, o processamento é reiniciado, dando continuidade ao ciclo. Assim, o serviço de replicação se mantém ativo, mesmo diante de falhas temporárias nas bases de dados. Quando uma base de dados é restabelecida, o serviço de replicação volta a atuar automaticamente. Apenas quando estas tentativas excedem um limite tolerável pré-estabelecido, o administrador do sistema deve ser notificado.

### 3.2.2. Versionamento

Uma base heterogênea versionada garante que o serviço de replicação manterá a consistência dos dados. Assim, se um módulo de uma base escrava possui a mesma versão do mesmo módulo na base mestre, então todas as atualizações geradas na base mestre foram devidamente aplicadas na base escrava.

Uma estratégia para se versionar os módulos é através da marcação de data e hora de cada atualização. Contudo, esta estratégia possui o inconveniente de que as bases mestres e escravas mantenham os relógios sincronizados. Sendo assim, optou-se por utilizar uma estratégia de versionamento baseada em contadores inteiros. Se um módulo está na versão X, deve ser aplicada uma atualização que o leva à versão X+1. O serviço de replicação é responsável por garantir que apenas atualizações que respeitem esta regra sejam aplicadas ao módulo de dados.

Para ilustrar esta regra, seja uma base com os módulos A, B e C com versionamento (200, 130, 500). Uma alteração «B,131» poderá ser efetuada e levará a base ao versionamento (200, 131, 500). Porém, se logo em seguida o serviço de replicação receber uma atualização «B, 133», ela será rejeitada, pois indica que houve perda ou atraso da mensagem «B, 132».

### 3.2.3. Controle dos Estados das Bases

Cada base de dados pode estar em um dos três estados distintos:

- **Mestre**: base que recebe alterações da aplicação
- **Escrava**: base que recebe alterações de serviços de replicação e é acessado pela aplicação apenas para leitura
- **Livre**: base que não está participando do ambiente de replicação

Com essa classificação, o serviço de replicação deve garantir que um fluxo de replicação sempre vá de uma base mestre para uma base escrava. Caso uma base mestre se perca, uma base escrava pode assumir o seu papel, através de um processo de *eleição*.



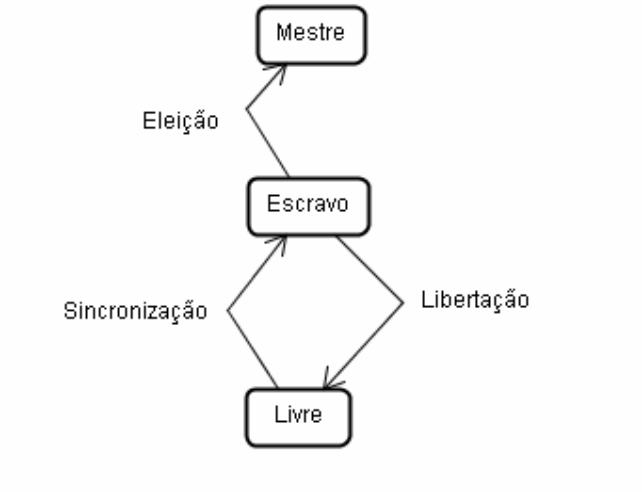


Figura 7. Possíveis trocas de estados de uma base

Se em um dado instante, um usuário deseje escrever em uma base escrava esta base deverá ter seu estado alterado para livre através de um *processo de libertação*. Com isso, esta base não está passível a receber atualizações vindas do fluxo de réplica e pode estar inconsistente com a base mestre. Assim, esta base necessitará ser *sincronizada* antes de retornar ao seu estado de escrava.

Os processos de libertação e sincronização são automatizados. Porém, são disparados somente quando são iniciados por analistas de suporte. Já o processo de eleição é definido e executado apenas por analistas de suporte e deve levar em

consideração questões como performance da rede, confiabilidade das atualizações, tolerância a atrasos e prioridades nas atualizações.

### 3.2.4. Coordenação por Multi-Serviços de Replicação

Foram analisadas três estratégias para coordenação do fluxo de replicação: um serviço centralizado, serviços ponto a ponto (P2P) e multi-serviços.

Na abordagem de um *serviço centralizado* (Figura 8a), tem-se um único responsável por realizar as replicações de todas as bases da topologia. Apesar da simplicidade de implementação, esta abordagem não explora o paralelismo existente entre as diversas bases de dados. Uma base é processada de cada vez. Para um grande número de bases, o atraso na replicação pode ser superior ao aceitável. Ou seja, esta solução não é escalável.

Já na estratégia de *serviços ponto a ponto* (Figura 8b), cada base teria seu respectivo serviço, separados fisicamente. Desta forma, pode-se explorar melhor o paralelismo do problema já que diversas bases independentes podem ser replicadas simultaneamente. Entretanto, esta abordagem envolve a instalação e manutenção de um serviço em cada ponto, encarecendo o processo de desenvolvimento e testes.

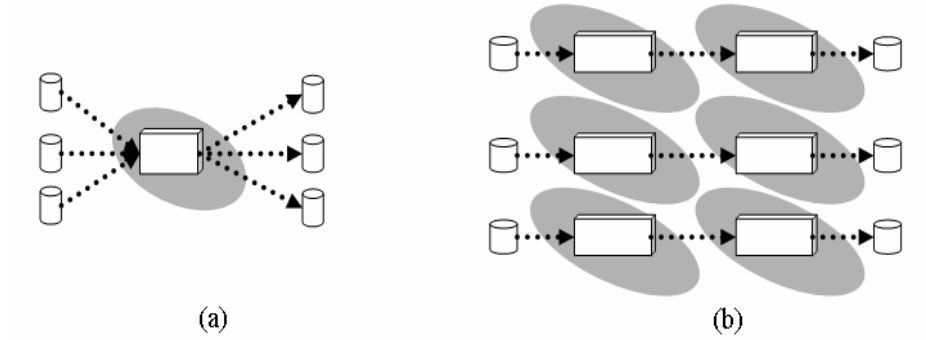


Figura 8. Estratégias de coordenação de fluxo de replicação

A arquitetura proposta neste trabalho segue uma nova abordagem onde a coordenação entre os fluxos é dividida em diversos nós lógicos, mantendo a proporção de um serviço para cada base. Contudo, os serviços podem estar distribuídos em um ou mais nós físicos. Assim, para um ambiente com **N** bases de dados, os serviços podem estar concentrados em um único servidor (Figura 9a) ou

distribuídos entre dois ou mais servidores (Figura 9b). É possível, inclusive, configurar **N** servidores distribuídos fisicamente. Esta arquitetura permite, então, um equilíbrio entre escalabilidade e custo de implementação.

Os serviços mantêm a comunicação via mensagens. Dessa maneira, as aplicações que acessam as bases de dados podem manter o seu trabalho independente do processamento de réplica. Embora este fluxo assíncrono permitir que as bases replicadas estejam temporariamente atrasadas em relação às bases originais, as aplicações não terão seu desempenho afetado pelo processo de replicação.

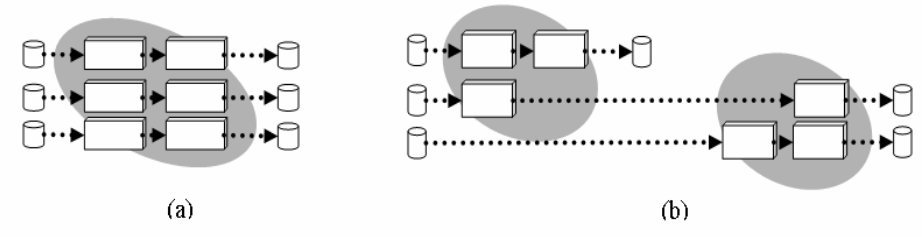


Figura 9. Possíveis configurações de uma topologia com multi-serviços

A tupla formada por uma base de dados e seu respectivo serviço de replicação foi denominada *nó de replicação* (Figura 10a). Um nó de replicação define a menor unidade de uma topologia de um ambiente de replicação (Figura 10b).

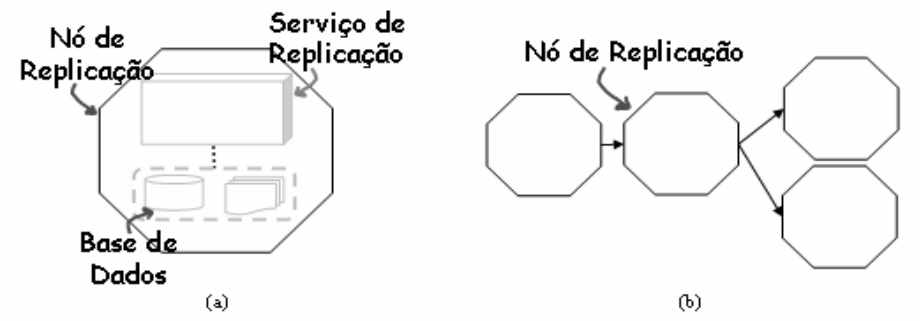


Figura 10. Nó e topologia

### 3.2.5. Processamento Assíncrono

O processamento de uma base de dados pode ser realizado de maneira assíncrona. Toda atualização gerada por uma aplicação é armazenada num registro de atualizações. Assim, a aplicação fica livre para realizar novas atualizações antes das antigas atualizações serem devidamente replicadas pelos serviços de replicação.

A comunicação entre serviços de replicação também pode ser realizada de maneira assíncrona, com troca de mensagens de atualização para suas respectivas bases de dados. Isto dá maior continuidade a um determinado serviço porque quando um serviço é interrompido e posteriormente restabelecido, basta continuar o processamento das mensagens restantes.



### 3.2.6. Operações Transacionais

As operações realizadas por um serviço de replicação devem ser realizadas transacionalmente. Cada alteração numa base de dados deve ser realizada transacionalmente para que em caso de falhas, a base se mantenha consistente.

Além disso, o processamento de mensagens de atualização também deve ser transacional para que em caso de falhas, a mensagem não seja perdida. Ou seja, é necessário manipular dois recursos numa única transação: a fila de mensagens e a base de dados.

Para realizar esta tarefa, é recomendado o uso do protocolo *two-phase commit* [BERNSTEIN, 1987] que garante que uma transação somente é finalizada com sucesso quando todos os recursos confirmam que a transação pode ser finalizada.

## 3.3. Componentes da Arquitetura

A arquitetura proposta neste trabalho define um conjunto de agentes e utilitários. Foram modelados três tipos de agentes:

- **Observador**: monitora as alterações da aplicação e as guarda num registro de alterações.

- **Exportador**: lê o registro de alterações de bases *mestres* e gera as mensagens adequadas.
- **Importador**: recebe mensagens dos exportadores e aplica as alterações nas bases *escravas*.

Um serviço de replicação é composto por um agente exportador e um agente importador. Durante o processo, o agente observador atua juntamente com a aplicação, monitorando as alterações na base de dados. O trabalho cooperativo entre estes agentes é suficiente para a realização de uma replicação.

Contudo, para facilitar o controle e a administração do ambiente foram implementados os seguintes utilitários: configurador, analisador, sincronizador e copiador. O copiador é responsável pela criação de bases escravas a partir de uma base mestre. Já o configurador é responsável pela manipulação dos serviços definidos para o ambiente de replicação. Por sua vez, o analisador possibilita verificar a consistência entre bases mestres e escravas, e também verifica a consistência entre módulos de uma mesma base de dados. Finalmente, o sincronizador permite povoar uma base mestre com alterações realizadas na base livre. A Figura 11 mostra uma visão geral destes agentes e utilitários. Nas próximas seções, eles serão detalhados.

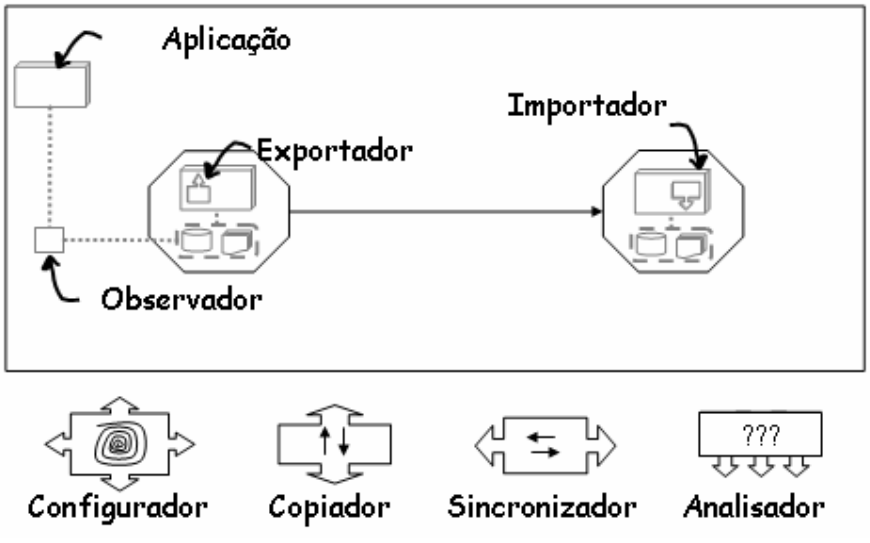


Figura 11. Visão geral da arquitetura e utilitários

### 3.3.1. Observador

Um agente observador intercepta as alterações do módulo de dados vindas da aplicação e armazena num registro de alterações. Para garantir que todas as alterações fossem mapeadas foi necessário identificar um ponto único por onde todas as alterações sempre passam independente da implementação da camada de persistência. O ponto único identificado foi a conexão com o banco de dados. Assim, foi definida uma implementação *proxy* de uma conexão com o banco. Este *proxy* aplica as alterações e também as armazena no registro de alterações, numa única transação. Por simplicidade e robustez, este registro se localiza fisicamente no próprio banco, em uma tabela específica (Figura 12).

Inicialmente o observador tenta executar as alterações no banco de dados através de um driver real. Em caso de sucesso, a instrução de alteração é inserida numa tabela do próprio banco. Assim, o serviço de replicação não precisa analisar os dados reais para identificar as alterações. Basta ler o registro de alterações. Embora seja transparente para aplicação, o observador deve ser distribuído em conjunto com a mesma.

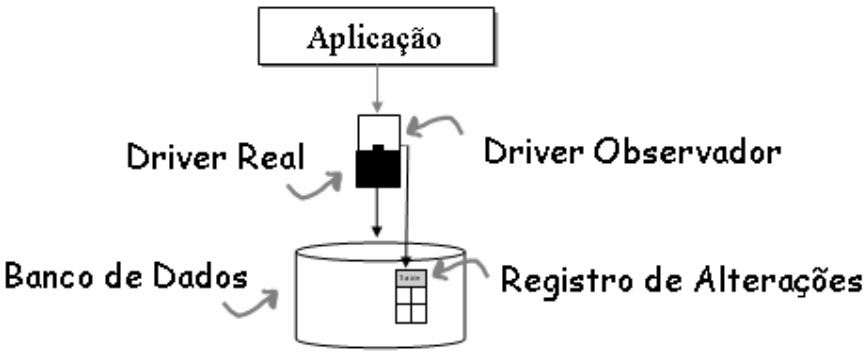


Figura 12. Driver observador

As alterações registradas pelo observador variam conforme o tipo de módulo de dados. Um módulo que armazena arquivos pode possuir alterações para criação, edição e remoção de arquivos. De forma equivalente, um módulo que é um banco de dados possui atualizações SQL e também de identificação de transações.

| Id | Instante | Tipo | Versão | Alteração |
|---|---|---|---|---|
| 152 | 12:31:05 | SQL | 345,20 | insert into ... |
| 153 | 12:31:08 | FILE | 345,20 | C:\temp\f.txt |
| 154 | 12:32:35 | SQL | 345,21 | delete from ... |
| 155 | 12:32:41 | COMMIT | 346,21 | - |

Figura 13. Exemplo de registro de alterações

A tabela da figura 13 ilustra um exemplo de registro de alterações. O campo *Id* se refere ao identificador de cada registro, garantindo unicidade. Em seguida, o campo *Instante* se refere ao instante no qual o registro foi efetuado. Este campo facilita a detecção de conflitos durante a sincronização de dados. Logo após, o campo *tipo* representa qual tipo de alteração está sendo registrada. Esta informação é importante para que o serviço de replicação saiba como a atualização deve ser aplicada na base escrava e como as versões devem ser verificadas. Já o campo *versão* informa o estado das versões de todos os módulos no instante onde a alteração aconteceu. Finalmente, o campo *alteração* armazena a atualização propriamente dita.

### 3.3.2. Exportador

Um agente *exportador* é responsável por verificar periodicamente uma base de dados. Caso existam alterações, ele deve gerar as mensagens equivalentes e enviá-las para uma fila. Em seguida deve apagar os registros de alterações para os quais já foram criadas as mensagens. Cada base mestre deve ser monitorada por apenas um exportador.

O exportador checa o registro de atualizações de uma base de dados e gera as mensagens de atualização. A leitura do registro é realizada em blocos transacionais. Assim, garante-se que as bases escravas mantenham a consistência transacional. Além disso, evita-se que a memória ocupada pelo exportador cresça indefinidamente.

Uma exportação é realizada pelas as seguintes etapas:

- **Leitura do bloco de atualizações.** Um bloco é delimitado pelo fim de uma transação. Caso mais de um módulo de dados seja atualizado

durante uma transação, serão registradas diversas atualizações com seus respectivos versionamentos.

- **Geração das mensagens.** As atualizações são encapsuladas em mensagens, adicionando-se um cabeçalho contendo informações de controle para o roteamento.
- **Envio das mensagens.** Após serem geradas, as mensagens são enviadas aos serviços que são responsáveis por aplicá-las. Estas mensagens são armazenadas em filas até que os importadores as processem.
- **Limpeza de atualizações processadas.** Caso as mensagens sejam enviadas com sucesso, as atualizações processadas devem ser eliminadas do registro de atualizações, para evitar mensagens duplicadas.

### 3.3.3. Importador

Um agente *importador* é responsável por coordenar atualização de uma base. Cada atualização é recebida através de mensagens vindas do exportador. Inicialmente importador lê a mensagem que está no topo da fila, realiza uma seqüência de verificações.

Uma importação é realizada pelas as seguintes etapas:

- **Leitura de uma mensagem.** O importador processa uma mensagem de cada vez. Assim, caso existam mensagens na fila, ele inicia o processamento da primeira mensagem colocada na fila.
- **Verificação da mensagem.** O importador verifica se a mensagem está devidamente endereçada, se a mensagem leva a versão do módulo atual da versão X para a versão X+1 e se o estado atual da base é escrava.
- **Aplicação da atualização.** As atualizações são aplicadas de acordo com o tipo de módulo. Um módulo que representa um banco de dados é atualizado através da execução de instruções SQL. Já um módulo que representa um conjunto de arquivos do sistema é atualizado através a inserção/alteração/remoção dos arquivos.

- **Reenvio da mensagem.** Caso haja uma replicação transitiva, ou seja, uma base **A** é replicada para uma base **B** que por sua vez é replicada para uma base **C**, uma mensagem que chega em **B** deve ser reenviada a **C**.
- **Agradecimento da mensagem.** Quando a atualização de uma mensagem é devidamente aplicada, a mensagem pode ser agradecida, que resulta na remoção da mensagem da fila. Dessa forma, o importador pode iniciar o processamento da próxima mensagem da fila.

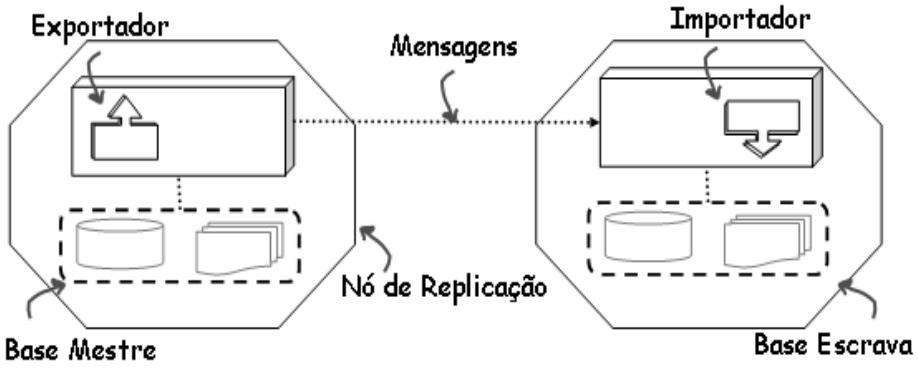


Figura 14. Serviços de replicação

### 3.5.1. Configurador

O configurador tem a missão de tornar fácil e ágil o processo de definição da topologia do ambiente de replicação bem como facilitar os controles de inicialização e finalização dos serviços. Através do configurador ainda é possível definir parâmetros de ajustes de desempenho para o ambiente em questão.

Por definição de topologia entende-se a declaração que quais são as bases de dados, quais são os módulos de cada base, e qual deverá ser o fluxo. Na declaração das bases e módulos de dados são definidos parâmetros de configuração de cada base. Já no fluxo são definidos quais módulos serão replicados, de qual base mestre para quais bases escravas. Ou seja, são definidos quais serão os módulos mestres e escravos.


Figura 15. Utilitário Configurador

Com a topologia definida, o configurador é capaz de criar e iniciar os serviços de replicação para cada base. Além disso, o configurador permite ajustar cada serviço de acordo com parâmetros como o intervalo para busca de atualizações, limite temporal para as transações, entre outros. Dessa forma, o usuário terá a possibilidade de conciliar desempenho do replicador com o desempenho da infra-estrutura de rede disponível.

### 3.5.2. Copiador

Para a criação de um fluxo de replicação são necessários no mínimo duas bases de dados. Uma será a base mestre e a outra escrava. Estas bases podem manter um número diferente de módulos, mas devem manter os mesmos módulos replicáveis. Por exemplo, se uma base mestre possui os módulos M1 e M2 e a base escrava possui os módulos M2 e M3, sendo que apenas o módulo M2 é replicável.

Dessa forma, antes da inicialização do serviço de replicação, é necessário que os módulos replicáveis estejam idênticos. Assim, no processo de geração da base escrava, devem-se copiar os módulos replicáveis da base mestre.

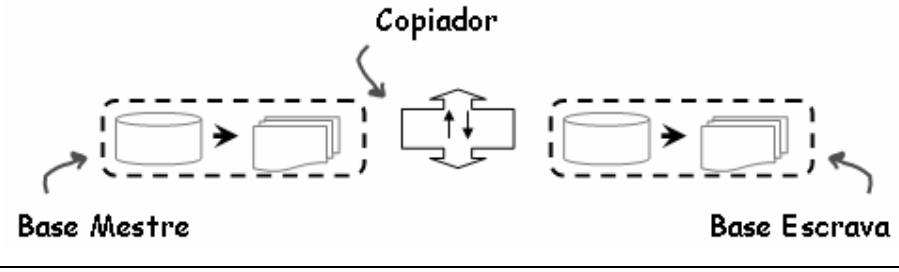


Figura 16. Utilitário copiador

O utilitário copiador é responsável por copiar os módulos replicáveis de uma base mestre para uma base escrava. Ele realiza um processo de congelamento do estado de uma base de dados. Esta versão congelada pode também ser utilizada como cópias de segurança, tecnicamente conhecidos como *backup/restore*.

### 3.5.3. Sincronizador

O utilitário sincronizador é responsável atualizar uma base mestre com dados vindos de uma base livre. Ele realiza trabalho semelhante ao serviço de replicação, realizando tarefas similares ao exportador e importador. Entretanto, o sincronizador difere na inexistência de troca de mensagens. Tanto a leitura quanto a execução de atualização ocorrem numa única transação distribuída. Consequentemente, a checagem de consistência também é diferente já que não há *roteamento* de mensagens. Além disso, num processo de sincronização há uma inversão nos papéis de num fluxo. A base mestre atua recebendo as atualizações de uma base livre, que era anteriormente escrava.

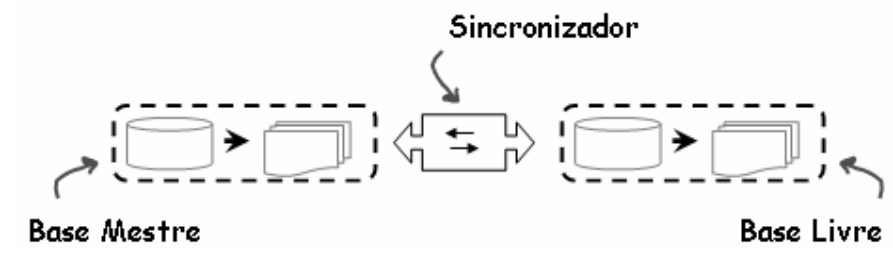


Figura 17. Utilitário sincronizador

### 3.5.4. Analisador

O utilitário analisador oferece a possibilidade de verificar de forma automatizada a consistência de toda a base de dados. Esta consistência inclui tanto a verificação de uma única base quanto a verificação comparativa entre uma base mestre e uma base escrava. Esta verificação é recomendada como pré-processamento na configuração dos serviços de replicação.

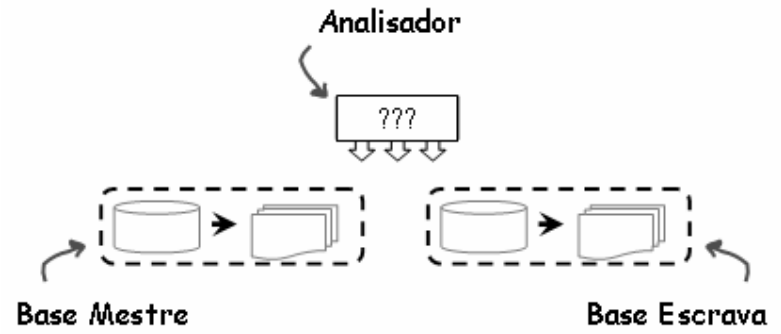


Figura 18. Utilitário analisador

Para verificar que os módulos de uma mesma base estão consistentes entre si, o analisador varre toda a base de dados buscando possíveis inconsistências nas referências entre os módulos. Por exemplo, se um banco de dados armazena o endereço de um arquivo, o analisador verifica que o arquivo existe no caminho indicado.

Já para verificar se as bases escravas estão consistentes com as bases mestre, o analisador primeiramente compara as versões de cada módulo. Caso estejam iguais, o analisador prossegue através de uma varredura comparativa.

Após o processamento, o analisador gera um relatório indicando as possíveis divergências entre o esperado e o verificado. As falhas de consistência locais indicam que houve manipulação indevida da base, possivelmente provocadas por erros nas aplicações que manipulam estas bases. Já as divergências entre uma base mestre e escrava indicam possíveis erros do serviço de replicação. Dessa forma, o analisador atua como suporte no diagnóstico de problemas.

## 3.6. Agentes, Utilitários e Soluções

As soluções para os problemas apresentados na seção 3.2 são alcançadas pela colaboração de um ou mais agentes e utilitários.

O processamento em ciclos é realizado por cada serviço de replicação, por seus agentes exportadores e importadores. Contudo, o configurador atua na definição do intervalo de duração de um ciclo e do intervalo entre um ciclo e outro.

O processamento assíncrono é resolvido em duas etapas. A primeira etapa consiste na comunicação entre um observador e um exportador e a segunda etapa ocorre entre um exportador e um importador. A comunicação entre um observador e um exportador é realizada através do registro de atualizações. E a comunicação entre um exportador e um importador é realizada via troca de mensagens.

As operações transacionais ocorrem no observador, no exportador, no importador e no sincronizador. No observador, o registro de atualizações ocorre na mesma transação onde a atualização está sendo realizada. No exportador, a leitura do registro de atualização, envio de mensagem e limpeza das atualizações processadas ocorre numa única transação. Já no importador, numa única transação é realizado o processamento de uma mensagem e a atualização da respectiva base. Finalmente, o sincronizador atua lendo as atualizações de uma base escrava e escrevendo numa base de mestre, numa única transação.

O versionamento é utilizado por todos os agentes e utilitários. Todos estes componentes da arquitetura avaliam a versão dos módulos das bases de dados quando fazem uma leitura e atualizam suas versões quando fazem uma escrita na base. Em especial, o configurador impede que os serviços de replicação se iniciem se as bases mestre e escrava não estiverem coerentes quanto ao versionamento.

O controle dos estados das bases é realizado pelo configurador e complementado pelo sincronizador. O configurador define quais bases são mestres e quais são escravas, de acordo com a definição da topologia. Já o sincronizador atua transformando uma base livre numa base escrava para se restabelecer o processo de replicação. Por fim, o agente importador atua verificando se possui acesso de escrita na base, de acordo com o seu estado.

Finalmente, a coordenação por multi-serviços é definida pela estrutura da arquitetura, onde o configurador atua gerenciando os serviços de replicação. Cada serviço de replicação possui um agente exportador e um agente importador.

| | **Agentes** | | | **Utilitários** | | | |
|---|---|---|---|---|---|---|---|
| | Observador | Exportador | Importador | Copiador | Configurador | Analisador | Sincronizador |
| Processamento em ciclos | | X | X | | X | | |
| Processamento Assíncrono | X | X | X | | X | | |
| Operações transacionais | X | X | X | | | | X |
| Versionamento | X | X | X | X | X | X | X |
| Controle dos estados das bases | | | X | | X | | X |
| Coordenação por multi-serviços | | X | X | | X | | |

Figura 19. Mapeamento entre as soluções e os agentes e utilitários

## 3.7. Camadas de um Serviço de Replicação

Seguindo a estratégia de projeto orientado a domínio, o serviço de replicação foi projetado utilizando uma arquitetura de camadas. Foram definidas três camadas:

- **Camada de Coordenação**: nesta camada estão localizados os agentes *exportador*, *importado* e *observadorr*.
- **Camada de Negócio:** esta camada define as regras de replicação para cada tipo de módulo de dados da base. Nela são definidos os tratamentos específicos para bancos de dados, arquivos e demais módulos de dados da base. Ela também especifica a direção dos fluxos de replicação.
- **Camada de Fronteira**: abstrai a comunicação de um serviço de replicação com os demais integrantes do ambiente.

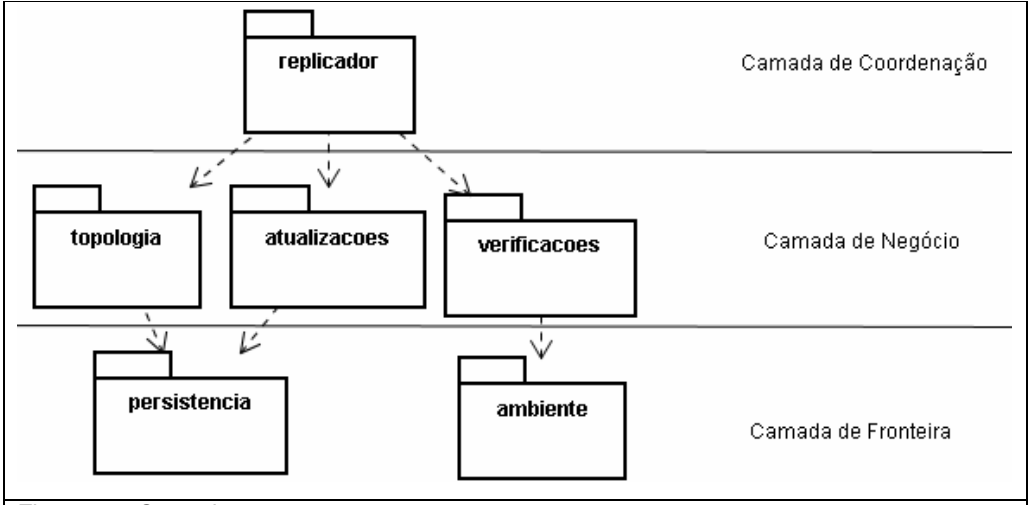


Figura 20. Camadas

Cada uma destas camadas será apresentada nas próximas seções.

### 3.7.1. Camada de Fronteira

Um serviço de replicação não possui uma camada de apresentação, por se tratar de uma aplicação que funciona em *background.* A interação humana ocorre apenas durante a configuração do serviço. Apesar disto, o serviço interage com outras aplicações através da camada de fronteira com o ambiente. Esta a camada é responsável por efetivar todas as comunicações com as bases de dados e com outros serviços de replicação.

As bases de dados são acessadas no pacote de persistência. Através dele, é possível ler as atualizações registradas pelo observador na base mestre e aplicá-las às bases escravas. Também é possível identificar qual é a versão de um módulo de uma base e alterá-la.

A camada de fronteira também possui o pacote de ambiente que implementa a comunicação assíncrona com outros serviços. Além disso, ela também abstrai questões relativas às tecnologias utilizadas para comunicação com as bases de dados.

### 3.7.2. Camada de Negócio

A camada de negócio formaliza as regras que os agentes devem obedecer ao replicar os módulos de dados. Este trabalho é dividido entre os pacotes *topologia*, *atualizações* e *verificações*.

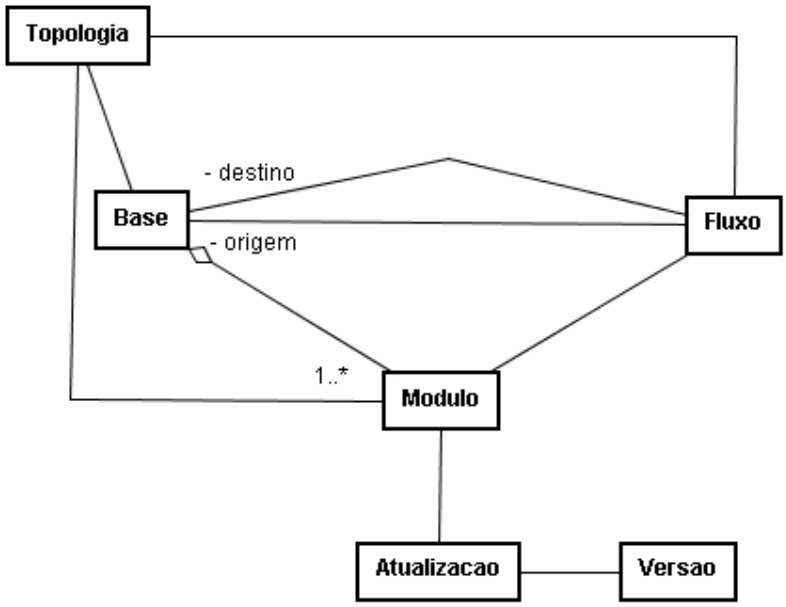


Figura 21. Classes do pacote topologia

O ambiente de replicação é descrito através do pacote *topologia*. Através desta descrição é possível responder qual módulo deve ser replicado de qual base mestre para qual base escrava. Esta orientação é fundamental para que os agentes dos serviços de replicação saibam quais são seus objetivos.

Já o pacote *atualizações* define o tratamento adequado para cada tipo de módulo de dados, além de informações de controle. As principais são:

- **SQL**: representa os comandos de inserção, atualização e remoção em bancos de dados.
- **FILE**: indica que um arquivo deve ser replicado.
- **COMMIT**: marca o final de uma transação

Finalmente, o pacote *verificações* da camada de negócio trata da validação de pré-condições para a aplicação das mensagens de atualização. Estas condições envolvem as estratégias de roteamento de fluxo, transações, algoritmos que garantem consistência, versionamento e performance. Assim, esta camada de

domínio concentra as regras de negócio e garante que o sistema tenha escalabilidade de requisitos.

### 3.7.3. Camada de Coordenação

Finalmente, a camada de coordenação define o agente exportador e o agente importador. Além disso, define as regras envolvidas com o ciclo de vida dos agentes. Este ciclo de vida envolve a criação dos agentes e define os ciclos de processamento.

Durante a criação, os agentes são informados sobre a topologia do ambiente. Assim, eles são capazes de identificar em quais bases devem atuar e com quais agentes devem se comunicar.

A camada de coordenação também provê mecanismos para controle de inicialização e finalização dos serviços de replicação. Estes recursos são acessados pelo utilitário configurador.