Representação de Áudio em Banco de Dados

Marisa Beck Figueiredo

Orientador

Prof. Dr. Caetano Traina Júnior

Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos - USP, como parte dos requisitos para a obtenção do título de Mestre junto à Ácrea de Ciências de Computação e Matemática Computacional

USP - São Carlos
Julho de 1997

"Transformar o medo em respeito, o respeito em confiança.
Descobrir como é bom chegar quando se tem paciência.
E para se chegar onde quer que seja, nao é preciso dominar
a força, mas a razão. É preciso antes de mais nada, querer."

Amyr Klink

Agradecimentos

À DEUS pelo amor e força dada nos momentos mais dolorosos. E pela alegria da conquista, pois sem ele nada seria possível.

À minha família que me apoiou desde o ínicio da minha caminhada, em especial à minha mãe que esteve ao meu lado nos piores momentos e às minhas irmãs Helena, Patrícia, Gisele, Rosangela e Nilsinho que fizeram parte da "torcida", fundamental para o meu mestrado.

Ao meu querido ex-orientador Fernando Traina, que sempre me incentivou e pela confiança no meu desempenho.

Ao meu digníssimo orientador Prof. Dr. Caetano Traina Júnior pelo total apoio, carinho e confiança, e pelos "preciosos" conselhos paternais. E à Prof. Dra. Agma J. M. Traina pela compreensão, apoio e "empréstimo" de sua sala para finalização da dissertação.

À CAPES pelo apoio financeiro.

Aos amigos do Grupo de Banco de Dados em especial ao Mauro e João que foram meus "mestres", ao Gazeta sempre solícito, e à Larissa que foi uma "irmã".

Ao Lacyr que colaborou com o desenvolvimento do trabalho.

Às professoras Rosely Sanches, Renata Pontin Fortes e Rosane Minghim pela "força" intelectual.

Aos amigos da USP que tive o prazer de conviver.

Às funcionárias da biblioteca que sempre demonstraram prestatividade, em especial à Silvana.

À Beth, Laura e Marília pela atenção dedicada.

Resumo

Os sistemas de banco de dados têm sofrido um processo de constante evolução, e ultimamente, diversas novas frentes têm se aberto. Uma frente importante é a da orientação a objetos, com a criação dos SGBDOO- Sistemas Gerenciadores de Banco de Dados Orientado a Objetos. Dentre os vários SGBDOO existentes temos: O_2, GemStone, Object Store, MRO-Modelo de Representação de Objetos, entre outros. Este trabalho apresenta um estudo de diversos sistemas. Em cada um deles foi analisado *interfaces*, controle de versões, gerenciamento de armazenamento, linguagem de manipulação de esquemas, arquitetura do sistema, entre outros.Além deste estudo foram analisadas características de áudio. De acordo com a acústica - uma parte da física - o **som** é o efeito produzido no aparelho auditivo pelas vibrações das moléculas de um meio transmissor. Este meio transmissor normalmente é o ar, mas também existem outros meios possíveis [Gome93].Foram verificadas as várias formas de gravação de áudio, entre elas a digitalização e a síntese. No processo de gravação foram analisados alguns formatos de arquivo para armazenamento, permitindo assim o balizamento da escolha. Dentre os formatos estudados estão: **MOD, WAV, SBI** e **SBK, SNG e MIDI**.Com base nestes estudos realizados, este trabalho apresenta uma avaliação das técnicas para a armazenagem e recuperação de áudio em bases de dados. O modelo de dados que serve de suporte para essa avaliação é o MRO sendo que o GEO é a implementação do MRO. Foi feito o tratamento de som em nível de esquema para o modelo e em nível de *interface* para o GEO. O tratamento de som permite o armazenamento e a recuperação de músicas, usando o formato MIDI.

Abstract

The database systems have undergone a process of constant evolution, and recently, several new possibilities have appeared. An important one is that of orientation to objects, with the creation of the DBMSOO-Database Manager Systems Oriented Object. Among the various existing DBMSOO, there are: O_2, GemStone, Object Store, MRO-Model Representation Objects and others. This work presents a study of several systems. In each one, *interfaces*, version control, storage management, scheme manipulation language, system architecture, among others, were analysed. Besides this study, audio characteristics were analysed. According to acoustic - a part of phisics - **sound** is the effect that is produced in the ear through vibration of molecules of a transmitting medium. This transmiting medium is usually the air, but there are also other possible media [Gome93]. The several processes of audio recording were verified, including the digitalization and the synthesis. In the recording process some fil formats for storage were analysed, what made the delimitation of the choice possible. Among the studied formats are: **MOD, WAV, SBI** e **SBK, SNG e MIDI**. Having these studies as a basis, this work presents an evaluation of the techniques for the audio atorage and recuperation in databases. The data modelo that serves as a support for this evaluation is the MRO, the GEO being the implementation of the ORM. The sound treatment was done in scheme level for the model and in interface level for the GEO. The sound treatmente allows the storage and recuperation of songs, using the MIDI format.

Índice

CAPÍTULO 1
Introdução

CAPÍTULO 2
Sistemas de Banco de Dados Orientado a Objetos

CAPÍTULO 3
Definições e Conceitos em Áudio

CAPÍTULO 4
Suporte de Áudio no MRO

CAPÍTULO 5
Implementação do Suporte de Partituras

CAPÍTULO 6
Conclusão

Figuras

Quadros

CAPÍTULO 1
Introdução

1.1 Considerações Iniciais

Este projeto é baseado em um SGBDOO-Sistema Gerenciador de Banco de Dados Orientado a Objetos chamado **GEO-GErenciador de Objetos**. O GEO tem sido construído no Instituto de Ciências Matemáticas de São Carlos com o objetivo de validação prática e estudo de como os conceitos de orientação a objetos podem ser implementados em um sistema de *software* real. Este gerenciador é baseado no modelo de dados denominado **Modelo de Representação de Objetos- MRO**. O principal objetivo de seu desenvolvimento é permitir a construção de SGBDs-Sistema Gerenciador de Banco de Dados e suportar a representação da informação dos sistemas de apoio a projetos de engenharia CAD-Computer Aided Design, de ambientes de suporte ao desenvolvimento de sistemas CASE-Computer Aided Software Engineering, e sistemas de coleta e tratamento de dados científicos/estatísticos.

As duas principais operações que devem ser suportadas eficiente e adequadamente por um SGBD são a consulta e a manipulação de dados. Em particular, em sistemas que suportam meta-estruturas, tal como é o caso do GEO, essas operações podem ser feitas tanto no esquema de dados quanto no banco de dados propriamente dita. O GEO, tal como definido pelo MRO, suporta a armazenagem e manipulação de dados com diversas características, as quais classificam os atributos que podem ser tratados. Todo atributo no GEO deve ter uma entre as seguintes características: propriedade, sinônimo, comentário, regra, procedimento, imagem, gráfico, visualização, tempo e estrutura [Biaj92].

A proposta deste trabalho é a inclusão no MRO (e implementação do respectivo suporte no GEO) de uma nova característica: áudio. O objetivo dessa característica é permitir a manipulação de sons como música e voz, entre outros, e sua definição deverá levar em conta essas modalidades de áudio. No entanto, devido à ampla abrangência desse tema, este projeto se restringirá a definir com detalhes apenas a representação de sons musicais. Foi construída também uma *interface*, permitindo a manipulação do áudio "música" no Gerenciador de Objetos, com o propósito de permitir avaliar a implementação desse suporte. Esta *interface* deverá permitir ao usuário a escolha de diversos tipos de sons, e suas características, tais como, intensidade, harmonia, timbre, entre outros.

1.2 Objetivos

As aplicações tradicionais de banco de dados incluem aqueles sistemas cuja estrutura dos dados tratados são intrinsicamente homogêneas. Normalmente, o que se tem são sistemas com um volume de dados relativamente grande, que são estruturados em uma pequena quantidade de estruturas simples. Um exemplo típico são os sistemas comerciais, em que todas as informações podem ser colocadas em algumas tabelas, cada uma com um volume relativamente grande de instâncias. Nesses sistemas os dados que podem ser armazenados, em particular os tipos de dados suportados, recaem nos usualmente suportados pelas linguagens tradicionais de programação, como números, datas, ou pequenas quantidades de texto - *strings*.

Os sistemas orientado a objetos caracterizam-se por suportar estruturas de dados bem mais elaboradas, além de permitir que a manipulação dessas estruturas seja também definida como parte delas próprias. No entanto, dentro das técnicas usuais da disciplina de banco de dados, é de praxe suportar ao nível do gerenciador de banco de dados, as operações usuais sobre os tipos de dados, que permitem a otimização das consultas sobre esses tipos de dados comuns às aplicações suportadas. Assim, o GEO define diversas características, cuja principal motivação é classificar tipos de dados não usuais quanto a operações que podem ser utilizadas para essa manipulação. Dessa forma, por exemplo, a característica de imagens refere-se a um tipo de dado que permite a armazenagem e manipulação de um atributo imagem. O conjunto de operações que podem ser definidas sobre imagens é completamente disjunto do conjunto de operações de busca e comparação de imagens (que pode e deve ser utilizado para implementar e otimizar as operações de busca do banco de dados). Tais operações são, portanto, distintas das que tradicionalmente são suportadas para os tipos mais convencionais de dados.

Neste contexto, este projeto tem como objetivo definir uma nova classe de tipos de dados, caracterizada como áudio, a qual pode ter as operações de busca e comparação adaptadas aos requisitos dessa classe. Para isso, é necessário especificar a nova classe de tipos de dado áudio, o que no MRO e no GEO é feito através da criação de uma nova característica de atributos.

Essa característica deve ser incluída entre os demais construtores semânticos do MRO e homogeneizada segundo os critérios de ortogonalidade de conceitos.

Dispondo desse recurso, é possível definir-se objetos no GEO que tenham como atributos, além

daqueles comumente utilizados tais como inteiros e *strings*, também atributos cujo valor é um som. Por exemplo, pode-se definir um objeto do tipo pássaro com os atributos do tipo *string* nome e país-de-origem, e os atributos do tipo áudio, como canto-de-chamada e sinal-de-perigo.

Vale ressaltar que este trabalho visa dar suporte dentro de um banco de dados, a informações não tradicionais, permitindo a armazenagem e recuperação de dados com a característica de áudio num banco de dados GEO. No entanto, este trabalho não se insere como pesquisa sobre o tratamento de informações multimídia, pois não leva em consideração o suporte à temporização e sincronização entre múltiplos atributos com características de áudio (ou mesmo imagens e gráficos).

Neste trabalho será necessário definir quais as operações que serão realizadas sobre o som. Para isso, é preciso definir uma estrutura que possuirá os atributos necessários para realizar as operações como, por exemplo, uma consulta a um determinado dado.

Inicialmente, será definido um esquema no qual existirá o tipo **som**, juntamente com seus atributos e relacionamentos. Na definição de atributos serão ainda definidas as possíveis restrições que esses possam ter. Através desta definição, será possível o armazenamento de som no MRO, permitindo a recuperação do som desejado. É importante esclarecer que, independente dos atributos criados, não será possível a recuperação de som através da sua reprodução. Por exemplo, considerando que esteja gravada a música "Eu sei que vou te amar" - Tom Jobim, esta não poderá ser recuperada pela gravação da música em PCM- Pulse Code Modulation, mesmo que seja com a voz de Tom Jobim, pois é muito difícil que as ondas sonoras gravadas na primeira vez sejam idênticas a uma outra. Por isso, a busca deverá ser feita através da partitura e instrumentos, que estarão ligados ao som. As músicas serão recuperadas de acordo com suas partituras, enquanto os intrumentos serão relacionados a seus parâmetros.

A **proposta deste trabalho** é permitir a representação de partituras de música no MRO e no GEO, que constitui a implementação do referido modelo. No MRO é necessário criar uma estrutura que defina a música, permitindo sua recuperação.

Usando o MRO, será definida toda a parte de esquema do banco de dados para inclusão de som. Neste modelo é feita a especificação de colônias, que são conjuntos de objetos de um mesmo tipo; objetos com seus respectivos relacionamentos e atributos, que podem ser classificados como: sinônimo,

comentário, regra, procedimento, tempo, estrutura de dados, gráfico, imagens, áudio e visualização.

É necessário incluir, entre os tipos de atributos, os tipos partitura de música e instrumento que permitirão a representação de som, mais especificamente de músicas.A partir disso, é criada uma estrutura para cada um dos respectivos tipos, permitindo assim a recuperação dos dados através do GEO. Ele é um gerenciador de objetos que foi baseado no MRO, permitindo a manipulação do esquema criado no MRO, e a inclusão de dados no banco de dados. Dessa forma, com a contribuição da pesquisa realizada neste mestrado também é implementado tratamento para áudio no GEO.

1.3 Organização do Trabalho

Este trabalho está dividido em seis capítulos. Cada capítulo aborda os seguintes aspectos:

o Capítulo 1 apresenta o trabalho, descrevendo seus objetivos e sua organização.No Capítulo 2 são descritos vários modelos de banco de dados, que foram estudados com o objetivo de verificar as diversas características existentes no banco de dados atuais. O tema áudio é abordado mais precisamente no Capítulo 3, onde são descritas sua definição, formas de representação e gravação, e o sistema MIDI.

O suporte de áudio no MRO é apresentado no Capítulo 4, definindo o meta-esquema e a extensão da LAMRO-Linguagem de Acesso ao MRO.O Capítulo 5 apresenta a implementação do suporte de partituras no GEO, descrevendo a arquitetura do GEO com suporte de partituras, além das operações de manipulação de partituras. Finalmente, no Capítulo 6, são apresentadas as conclusões deste trabalho.

CAPÍTULO 2
Sistemas de Banco de Dados Orientado a Objetos

Neste capítulo são descritos os Sistemas de Banco de Dados Orientado a Objetos, visando apresentar o estudo realizado durante a revisão bibliográfica. Os sistemas revisados foram: O_2, ObjectStore, GemStone, MRO, os quais são descritos nas próximas seções.

2.1 O Sistema O_2

O sistema O_2 é descrito nesta seção, seguindo a subdivisão:

- História e Situação Atual
- Arquitetura do Sistema

dando uma visão específica relacionada ao trabalho proposto.

2.1.1 História e Situação Atual

O sistema O_2 foi projetado e desenvolvido por um consórcio de pesquisa, composto pelo INRIA-Institut National de la Recherche Informatique et Automatique, Siemens - Nixdorf, Bull, o CNRS-Centre National de la Recherche Scientifique e a Universidade de Paris. Esse projeto iniciou-se em 1986 e seu objetivo era projetar e implementar um sistema de banco de dados da geração seguinte.

Após várias tentativas com protótipos, foram feitas melhorias tanto através da inclusão de novas características, como pela construção de um código mais robusto. Uma versão completa do sistema foi testada em 1990. No final de 1990, foi criada a companhia O_2 Technology, e ficou responsável pelo desenvolvimento, manutenção e divulgação do produto. A venda comercial iniciou-se em junho de 1991 [Deux91].

As áreas de aplicação adequadas ao O_2 incluem as "novas aplicações", tais como CAD/CAM-Computer Aided Design/Computer Aided Manufacturing, sistemas urbanos e geográficos, sistemas de informação editorial, automação de escritório e aplicações comerciais.O produto O_2 possui três objetivos principais:

- aumentar a produtividade de desenvolvimento de aplicação tanto para aplicações tradicionais como para as atuais;

- fornecer ferramentas que permitam o desenvolvimento de novas aplicações;
- melhorar a qualidade de aplicações finais.

Para alcançar estes objetivos, o O_2 fundamenta em três idéias:

- a união de *interfaces* de usuário, linguagens de programação e tecnologias de banco de dados;
- o uso de tecnologia orientada a objetos;
- a aderência a padrões [Deux91].

Tais idéias abrangeram desde a aplicação de conceitos orientado a objetos até a portabilidade, possuindo as seguintes vantagens:

- oferece maior produtividade durante as fases de desenvolvimento de aplicação do projeto, codificação, teste e manutenção;
- gera aplicações de alto desempenho;
- produz aplicações gráficas de alta qualidade;
- suporta aplicações novas, tais como CASE, GIS-Geografic Information System, CAD/CAM, multimídia, gerenciamento de documentação e dados técnicos, gerenciamento de rede assim como as aplicações de tipo tradicional;
- reusa as aplicações existentes e faz melhorias por um custo mínimo [Banc92];

Portanto, permite um suporte bastante.

2.1.2 A Arquitetura do Sistema

A **arquitetura do sistema** é organizada em três níveis, vista na Figura 2.1. O nível mais alto é o do **Gerenciador de Esquema** como mostrado na Figura 2.1. As funções fornecidas por ele incluem criação, acesso, modificação e remoção de classes, métodos e variáveis globais. Além disso, ele é responsável por controlar a consistência do esquema e pela checagem das regras do subtipo em hierarquias de herança.

O nível intermediário é o do **Gerenciador de Objeto**. Este componente gerencia objetos e valores complexos independente de sua persistência. O gerenciador suporta troca de mensagem e configuração cliente/servidor. Além disso, ele implementa todas as funções relacionadas à persistência,

coleta de lixo e mecanismos de acesso, tais como índices. Finalmente, ele fornece todas as funções para gerenciamento de transação.

O nível mais baixo é o **WiSS**-Wisconsin Management Subsystem, que gerencia o armazenamento secundário. O WiSS possui a funcionalidade para as atividades de persistência, gerenciamento de disco e controle de concorrência para registros [Bert93].

O O_2 pode suportar dois tipos de ***interfaces***: ***interface* de linguagens** e o **ambiente O_2** .

A *interface* de linguagens permite que um programa escrito em C ou C++ obtenha vantagens dos serviços do O_2, declarando esquemas O_2 e armazenamento, propagação e manipulação em banco de dados O_2 . Alternativamente, o usuário pode se beneficiar de todo o ambiente O_2 . Este ambiente inclui:

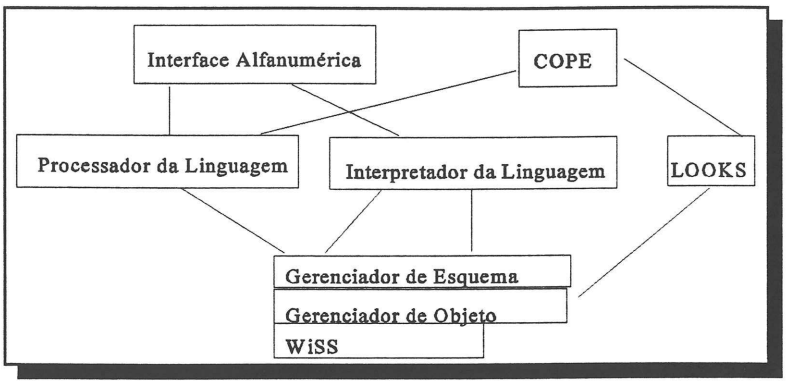

Figura 2.1 - Arquitetura do O_2

- uma linguagem de consulta, O_2Query;
- um gerador de *interface* de usuário, O_2Look;
- uma linguagem de quarta geração, O_2C;
- um ambiente de programação gráfica incluindo um depurador e um *browser* de banco de dados e esquema.

A seguir serão brevemente descritos os componentes do ambiente O_2, segundo [Deux91].

2.1.2.1 Linguagem de Consulta

A **O_2Query** é uma linguagem de consulta estendida, parecida com SQL-Structured Query Language, que trabalha com valores e objetos complexos. É um subconjunto de O_2C, mas pode ser usada independentemente como uma linguagem de consulta interativa inata ou como uma função chamada pela linguagem C ou C++ [Deux91].

Todos os operadores e tipos de dados O_2, incluindo os métodos, são permitidos dentro de uma consulta. A parte ***select*** de uma consulta define a estrutura do resultado. A parte ***from*** introduz os conjuntos e listas pelos quais a consulta será executada. Finalmente, a cláusula ***where*** introduz um predicado que filtra os resultados [Deux91]. Uma visão geral do gerador de *interface* O_2Look é apresentada em seguida.

2.1.2.2 O Gerador de *Interface* do Usuário O_2Look

O gerador de *interface* do usuário O_2Look suporta a exibição e manipulação na tela de objetos multimídia volumosos e complexos. O O_2 soluciona dois problemas:

- fornece ao programador facilidades para criação de *interfaces* gráficas de alta qualidade, por meio das apresentações do objeto *ready made*, sobre a tela;
- as apresentações *ready made* podem ser usadas para unir os requisitos de aplicações específicos. Por exemplo, cores, tipos, fontes e *layouts* podem ser redefinidos [Deux91].

O O_2Look proporciona funções para manipular apresentações de valores e objetos O_2. O O_2Look é implementado em C++ sobre o Motif e o Sistema XWindow.

As **características principais** do O_2Look são:

- **Apresentações *ready made***

Independentemente da complexidade dos dados, o O_2Look pode construir uma apresentação *ready made* de um objeto. Um menu *pop-up* permite que o usuário ative qualquer um dos seus métodos. Uma apresentação pode ser editada. Quando a edição é completada, o objeto pode ser reescrito no banco de dados [Deux91].

- **Cortar/Copiar e Colar**

As apresentações podem ser editadas. Tradicionalmente, as operações cortar, copiar e colar se referem ao texto todo. Em O_2Look, estas operações aplicam-se a estruturas. O O_2Look realiza verificação automática do tipo dinâmico.

•Máscaras e Recursos

As máscaras e recursos são usados para utilizar apresentações *ready made*. Uma máscara é uma estrutura usada para expressar quais as partes de um objeto que deveriam ser visíveis e editáveis. Uma máscara é também usada para chamar os componentes gráficos constituindo uma apresentação. Os nomes são usados para mapear um *widget* para recursos gráficos. Um recurso pode ser qualquer parâmetro gráfico de um *widget*: cor de fundo, fontes de título, *layout*, itens de menu, entre outros. Os arquivos de recursos são usados para descrever o mapeamento entre *widgets* e recursos. Em razão dos arquivos de recursos serem interpretados em tempo de execução, o O_2Look permite o uso de *interface* de usuário de uma aplicação sem recompilação. Uma dada aplicação pode ter um arquivo de recurso em inglês ou francês, e utilizar um ou outro instantaneamente [Deux91].

2.1.2.3 Suporte a Linguagem e Modelo de Dados

Em O_2 , um **sistema** consiste de um conjunto de esquemas e banco de dados. Um esquema define tipos de dados, enquanto um banco de dados contém o próprio dado.

O **O_2C** é uma linguagem de quarta geração, pois permite que o usuário realize três tarefas: programação, manipulação de banco de dados e geração de *interface* do usuário.

O O_2 manipula valores e objetos. Um valor tem um tipo. Este tipo é recursivamente definido a partir de tipos atômicos e construtores de tipos. Um objeto tem uma identidade, um valor e um comportamento, definido por seus métodos. Portanto, o objeto pertence a uma classe.

Um objeto ou valor pode fazer referência a outro objeto via suas identidades. Similarmente, um objeto ou valor pode ser composto de subvalores. Em último caso, entretanto, um subvalor é parte de seu todo e não pode ser compartilhado por outros objetos. Em outras palavras, atribuir um valor gera uma operação de cópia, enquanto atribuir um objeto resulta em uma referência nova para este objeto. Os **tipos atômicos** são: booleano, caracter, inteiro, real, *string* e *bits*. A classe é definida por seu tipo e seu método, e possui objetos [Deux91]. A *interface* do usuário é apresentada a seguir.

2.1.2.4 Interação Usuário Final em um Ambiente de Banco de Dados

Muitas ferramentas de *interface* de usuário estão disponíveis atualmente, mas as necessidades dos programadores de banco de dados são específicas e não são direcionadas por ferramentas clássicas. A particularidade na construção da *interface* do usuário em um ambiente de banco de dados, encontra-se na necessidade de exibir, editar e manipular diretamente objetos complexos e multimídia [Deux91].

2.2 Sistema ObjectStore

O Sistema **ObjectStore** é definido e caracterizado juntamente com seus objetivos. Além disso, é descrita sua *interface* e o suporte a versões.

2.2.1 Características do ObjectStore

O **ObjectStore** é um SGBDOO-Sistema Gerenciador de Banco de Dados Orientado a Objetos de alto desempenho, projetado para facilitar o uso em aplicações sofisticadas usando técnicas de desenvolvimento orientado a objetos. Ele oferece uma *interface* de linguagem altamente integrada para um conjunto completo de características, de sistemas gerenciadores de banco de dados tradicionais incluindo persistência, gerenciamento de transação e acesso distribuído. As facilidades do gerenciamento de dados, combinadas com as ferramentas de desenvolvimento, criam um ambiente de desenvolvimento altamente produtivo para implementar aplicações orientada a objetos [Obje95].

Dentre as características mais importantes, temos [Obje95]:

- *interface* transparente projetada para ambientes de programação C ou C++;
- acesso concorrente a grandes quantidades de dados persistentes;
- distribuição de objetos na rede, usando uma variedade de protocolos de redes populares;
- acesso a dados persistentes na mesma velocidade que dados transientes;
- capacidade de modelagem de dados extensíveis para aplicações com estruturas de dados complexas;
- facilidade de migração de caminho para aplicações em C ou C++;
- bibliotecas de classes para gerenciamento de versão e configuração;
- bibliotecas de classes para gerenciar coleções de objetos;

- protocolo meta-objeto com acesso programático a informações do esquema.

Estas características, por sua vez, possibilitam desde um acesso concorrente até características de portabilidade.

2.2.2 Vantagens

O modelo de distribuição do ObjectStore é extremamente flexível; ele trata o cliente e servidor como processos que podem ser executados em qualquer lugar desejado. O objetivo do **ObjectStore** é assegurar que quando os dados são solicitados pela aplicação, estes ou estão na memória virtual da aplicação, ou são levados rapidamente para a memória [Obje95].

Dentre suas **vantagens**, temos:

- armazenamento, distribuição e gerenciamento de todos os formatos de dados no seu banco de dados. Ele permite armazenamento direto de relacionamentos, em particular, relacionamentos multi-valorados. Com ele, é possível modelar uma rede de telefones ou até um sistema financeiro;

- o suporte a *groupware* fornece aos usuários várias versões de seus objetos de aplicação e permite o agrupamento daqueles objetos arbitrariamente. O ObjectStore suporta aplicações *groupware* durante transações curtas e longas.

2.2.3 Objetivos do ObjectStore

Dentre os **objetivos principais**, temos:

- desempenho para dados sofisticados;
- custo efetivo de desenvolvimento;
- alinhamento rápido.

O ObjectStore representa um avanço revolucionário no gerenciamento de dados. A arquitetura oferece inovações nas seguintes áreas [Obje95]:

- alto desempenho ou aplicações interativas;
- facilidade de uso para programação orientada a objetos;

- migração de código de aplicações existentes;
- capacidade de modelagem de dados flexíveis;

A seguir, são explicados com mais detalhes alguns destes aspectos inovativos do ObjectStore.

2.2.3.1 Alto Desempenho

O primeiro objetivo do ObjectStore é o de fornecer alto desempenho de banco de dados para aplicações tais como gerenciamento de rede, projeto de engenharia, e automação de processos, que têm os dados armazenados e acessados como objetos. Esses tipos de aplicações diferenciam-se das aplicações tradicionais de banco de dados, pois tratam com modelos de dados complexos e executam pequenas operações de banco de dados com uma quantidade menor de processamento. Se toda operação do banco de dados necessitar de um custo de *overhead* por operação, o custo do *overhead* será exorbitante. É fundamental que a manipulação de dados armazenados sejam tão rápida quanto possível. O ObjectStore é projetado para minimizar o *overhead* para [Obje95]:

- **acesso a rede** - já que os acessos à rede são operações dispendiosas, reduzir o número de transferência de dados é crucial para o sucesso de sistemas gerenciadores de banco de dados distribuídos. O ObjectStore reune objetos a serem transmitidos entre cliente e servidor, e somente um pedido da rede é usado para enviar um grupo de objetos referenciados no *cache* do cliente. Esta estratégia minimiza o tráfego da rede, e permite que o cliente continue uma transação sem comunicar com o servidor até que a transação seja finalizada [Obje95].

- ***caching* de dados** - mesmo quando muitos usuários têm acesso a um banco de dados compartilhado, muito freqüentemente o próximo usuário de um dado será o mesmo que o usuário anterior. Em outras palavras, embora o acesso concorrente deva ser concedido e funcionar corretamente, muitos dados deverão ser usados principalmente por um usuário em um curto espaço de tempo. O ObjectStore fornece um mecanismo de *caching* que concede ao usuário um novo acesso a objetos já buscados anteriormente, sem causar o *overhead* para *refetching* ou *relocking* daqueles objetos. Este *caching* de dados significa que quando uma seqüência de transações acessam os mesmos objetos, existe uma alta probabilidade que os dados acessados na próxima transação ainda sejam colocados na memória da estação de trabalho [Obje95].

• **acesso a disco** - freqüentemente uma aplicação usa somente uma parte do banco de dados, e aquela parte do *overhead* é armazenada continuamente numa seção menor do banco de dados. Com a facilidade de agrupamento do ObjectStore, os objetos relacionados podem ser agrupados juntamente no banco de dados, acentuando a referência de localidade. Os projetistas do ObjectStore podem alocar objetos em um banco de dados, em um segmento específico do banco de dados, ou em um objeto específico [Obje95].

2.2.3.2 Facilidade de Uso

A "chave" para a facilidade de uso, principalmente usando a programação, está na integração entre o sistema banco de dados e a linguagem de programação da máquina. O ObjectStore usa uma técnica para armazenar objetos nos quais a persistência não é parte do tipo de um objeto (isto é, a persistência é independente do tipo). Por exemplo, os objetos de qualquer tipo de dado C++ ou Smalltalk podem ser alocados por tipos básicos tais como: inteiros ou *strings* de caracter de modo transiente ou persistente em estruturas definidas pelo usuário [Obje95]. Esta facilidade resulta em algumas vantagens, entre elas:

• **Facilidade de Entendimento**

Os desenvolvedores, que usam orientação a objetos, já têm o conhecimento básico para usar ObjectStore, pois escrever uma aplicação que usa ObjectStore é como escrever um programa C++ comum. Tipos, variáveis e classes são declaradas da mesma forma. Similarmente, a manipulação, o acesso a objetos e as chamadas de funções são gerenciadas da mesma maneira [Obje95].

• **Reusabilidade**

O ObjectStore permite que o mesmo código opere em dados persistentes ou transientes, e ainda, que as bibliotecas, desenvolvidas para manipular dados transientes trabalhem com dados persistentes freqüentemente sem alterações.

Normalmente, se uma biblioteca não precisa fazer qualquer alocação persistente de si própria, ela pode ser aplicada a dados persistentes sem ser recompilada. Por exemplo, uma biblioteca de rotina

que tem um vetor de números ponto-flutuante e calcula um algoritmo complexo, este pode ser passada para um vetor persistente ObjectStore. Isto significa que os usuários, que tem investimento nas bibliotecas de classes podem continuar usando aquelas bibliotecas sem ter que criar versões específicas para manter dados persistentes [Obje95].

2.2.4 Aplicação da *Interface*

O ObjectStore fornece suporte a acesso de dados persistentes em transações, uma biblioteca de coleção de tipos, relacionamentos bidirecionais, uma facilidade de consulta, e uma facilidade de versão para suportar trabalho colaborativo.

Existem três *interfaces* de programação: uma *interface* da biblioteca C, uma *interface* da biblioteca C++ e um C++ estendido que fornece um compactador de integração da linguagem para facilidades de consulta e relacionamento [Lamb91], permitindo assim uma boa abrangência de linguagens para o usuário.

2.2.5 Versões

O ObjectStore fornece facilidades para que vários usuários compartilhem dados de uma forma cooperativa. Através destas facilidades, um usuário pode avaliar uma versão de um objeto ou grupo de objetos, fazer alterações e verificar mudanças anteriores ao projeto de desenvolvimento principal para que sejam visíveis a outros membros do conjunto cooperativo. Neste interim, outros usuários podem continuar a usar as versões anteriores; logo, não são impedidos por conflitos de concorrência nos dados compartilhados, apesar da duração das sessões de edição envolvida [Lamb91].

Se outros usuários querem fazer algumas alterações, eles verificam versões alternativas do mesmo objeto ou grupos de objetos, e trabalham sobre suas versões em particular. Novamente, o resultado é que não existem conflitos de concorrência, mesmo que os usuários estejam operando sobre os mesmo objetos. As versões alternativas podem posteriormente ser fundidas para reconciliar as diferenças resultantes deste desenvolvimento paralelo. Esta operação de junção é um problema difícil e é deixado para o usuário implementar sob uma aplicação específica. Em suporte a isto, o ObjectStore permite acesso simultâneo em ambas as versões de um objeto, durante a junção.

Os usuários podem controlar as versões usadas, para cada objeto ou grupo de objetos de interesse, definindo espaços de trabalho privados que especificam a versão desejada. Esta pode ser a versão mais recente, a versão anterior ou até mesmo uma parte da versão [Lamb91]. Em seguida é descrita uma visão geral da arquitetura do ObjectStore.

2.2.6 Arquitetura

Uma operação fundamental de uma linguagem de programação de banco de dados é a **dereferenciação**: encontrar e usar um objeto destino que é referenciado por um objeto origem [Lamb91]. O objetivo da *interface* do ObjectStore estabelece que esta deve ser como um programa C++ comum, para fornecer integração transparente com a linguagem e tornar a dereferenciação tão rápida quanto possível. Isto significa que os ponteiros de uma linguagem de máquina devem ser capazes de servir como referências a um objeto persistente.

O servidor do ObjectStore fornece um depósito de período longo para dados persistentes. O banco de dados pode ser armazenado de duas maneiras: dentro de arquivos produzidos pelo sistema ou de arquivo do próprio ObjectStore. O servidor e o cliente se comunicam via rede de área local, quando estão rodando em máquinas diferentes, e por facilidades mais rápidas, tais como, memória compartilhada, quando estão rodando na mesma máquina.

O servidor armazena e recupera páginas de dados em resposta a pedidos de clientes. O servidor não tem conhecimento do conteúdo de uma página. Ele simplesmente transporta páginas para/de clientes, e as armazena no disco. O servidor é responsável também pelo controle de concorrência e recuperação, usando técnicas semelhantes àquelas usadas em SGBDs tradicionais.

Em razão do servidor não ter conhecimento do conteúdo da página, muitas das consultas são feitas pelo cliente.

O ObjectStore mantém um *cache* cliente, um grupo de página de banco de dados que foram recentemente usadas, na memória virtual da máquina cliente. Quando a aplicação sinaliza uma falha na memória, o ObjectStore determina se a página que está sendo acessada está no *cache* cliente. Se não, este solicita ao servidor do ObjectStore a transmissão da página para o cliente, e coloca a página no *cache* cliente. Então, a página do *cache* cliente é mapeada em espaço de endereço virtual, para que a

aplicação possa acessá-la. Finalmente, a instrução defeituosa é reinicializada, e a aplicação continua [Lamb91], finalizando assim o processo. Em seguida, o modelo GemStone é descrito, dando uma visão geral de seus conceitos.

2.3 O Modelo GemStone

O Modelo GemStone será descrito em detalhes a seguir. Além disso, será apresentada uma descrição de sua arquitetura, características do sistema, *interface*, juntamente com as várias modificações existentes no sistema.

2.3.1 Características do GemStone

O sistema **GemStone** foi desenvolvido pela Corporação de Desenvolvimento ServioLogic com o objetivo de fornecer um sistema de gerenciamento de banco de dados caracterizado por um forte modelo de dados e, portanto, reduzir o tempo necessário para desenvolvimento de aplicações complexas.

O **GemStone** é um SGBDOO, que compartilha os conceitos da linguagem de programação orientada a objetos Smalltalk com as funções de um sistema de gerenciamento de dados [Bert93]. A linguagem de definição e manipulação de dados é chamada OPAL e é derivada de Smalltalk. Adotando a filosofia de Smalltalk, cada entidade no sistema, incluindo os programas escritos em OPAL, são considerados objetos.

No GemStone, os métodos e as estruturas comuns a todas as instâncias de uma classe estão contidas em um objeto chamado ODC - Objeto Definido pela Classe. Por isso, até as definições de classes estão em objetos. Todas as instâncias de uma classe contém uma referência a seu ODC como parte do identificador do objeto. Além disso, cada objeto é a instância de uma classe precisamente. A estrutura interna de muitos dos objetos consistem de vários atributos que podem ter valores e referências a outros objetos. Os objetos estão organizados por meio de estruturas, que são obtidas pela combinação de quatro formatos de armazenamento básico, segundo [Bert93]:

- **atômica** - esses são objetos tais como inteiros e *strings*, que não tem estrutura interna;
- **variáveis de instância** - essas são unidades de armazenamento, que podem ser classificadas por nome;

• **variáveis de instância indexável** - são unidades de armazenamento classificadas por número. Um exemplo é a classe *Array*;

• **variáveis de instância anônima** - essas diferem das últimas duas, pois são acessadas pelo nome mais do que pelo valor. As instâncias da classe *Set* pertence a esta categoria [Bert93].

Dentre as características encontradas no GemStone, temos:

• **suporte concorrente para várias linguagens** - GemStone fornece suporte concorrente para aplicações desenvolvidas em Smalltalk, C++ ou C. Todas as aplicações, apesar da linguagem, podem ter acesso simultâneo aos mesmos objetos do banco de dados;

• **controle de Transação Multi-Usuário** - vários usuários podem operar simultaneamente no banco de dados, com uma variedade de modos de controle de transação disponíveis;

• **segurança em Nível de Objeto** - Controle de autorização podem ser aplicados para qualquer objeto no banco de dados [Serv87].

São apresentadas a seguir as características principais do modelo e uma visão geral da arquitetura do sistema.

2.3.2 A Arquitetura do Sistema

A arquitetura do GemStone inclui dois processos, chamados **Gem** e **Stone** [Catt94]. O **processo Stone** é o gerenciador de dados, fornecendo entrada/saída de disco, controle de concorrência, autorização, transações e recuperação. O Stone reside na máquina servidora, acessando o disco através das chamadas do sistema operacional.

O **processo Gem** fornece compilação dos programas OPAL e autorização do usuário. O processo Gem pode residir na estação de trabalho servidor ou cliente. Além disso, existe uma comunicação entre processos, possivelmente sobre uma rede, entre o usuário do programa, o processo Gem, o processo Stone e o sistema operacional, nesta ordem. O processo Gem pode buscar, fechar e esconder objetos, páginas, ou segmentos completos de dados por vez em uma rede, já que este pode ser otimizado para as necessidades de uma aplicação [Catt94].

Ele tem uma arquitetura distribuída, como mostrada na Figura 2.2, e consiste de um conjunto de PC-IBM e/ou estações de trabalho Smalltalk-80 e de um servidor de objetos, implementado no sistema

de arquivo VAX/VMS, conectado através de uma rede local.

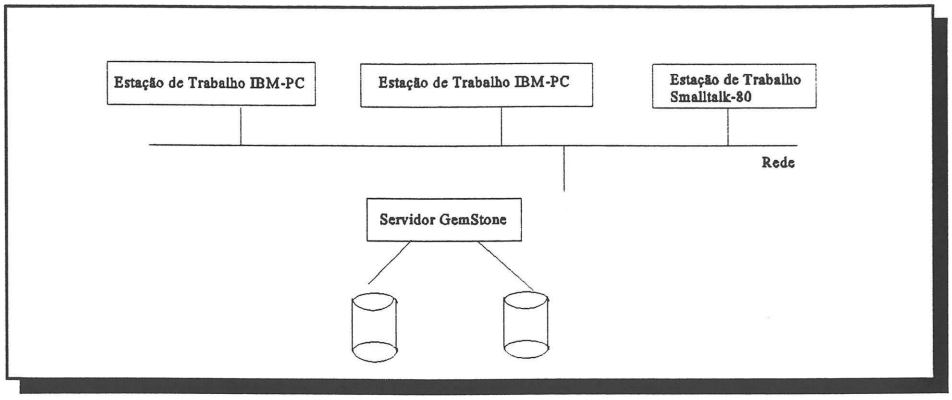

Figura 2.2 - Arquitetura do GemStone

O modelo GemStone é baseado nos conceitos de objeto, classe e mensagem. As classes são organizadas em hierarquias com herança simples. As aplicações podem ser escritas em linguagens como: OPAL (extensão de Smalltalk), C, C++ e Pascal.

O APO-Ambiente de Programação OPAL é um conjunto de aplicações, compatíveis com o Microsoft Windows, mostradas na Figura 2.3, entre elas:

- uma **classe *browser*** que permite ao usuário, examinar, adicionar e/ou modificar definições de classes GemStone;
- um poderoso ***loader/dumper***, com que o usuário pode transferir dados formatados (registros de tamanho fixo) entre arquivos PC e o servidor GemStone;
- um **editor *workspace***, com o qual o usuário pode criar, editar e executar expressões OPAL.

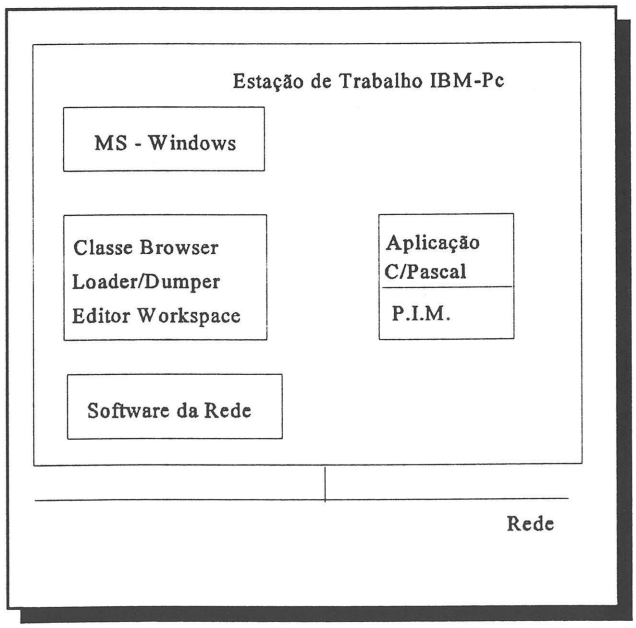

Figura 2.3 - Ambiente de Programação OPAL

Para suportar aplicações escritas em linguagem procedural, o sistema fornece módulos

de objeto chamados MIP-Módulos de *Interface* Procedural, no qual o usuário "fecha" as aplicações que são executadas no PC-Personal Computer [Bert93]. Estes módulos implementam chamadas de procedimento remoto para a função fornecida pelo Servidor GemStone.

No servidor, os dois componentes podem ser identificados, como mostrado na Figura 2.4, segundo [Serv87]:

- O **Gem** implementa a memória do objeto e a "máquina virtual" padrão Smalltalk. O Gem compila e executa métodos escritos em OPAL e gerencia o controle da sessão e autenticação;
- O **Stone** fornece o gerenciamento de armazenamento secundário, controle de concorrência, autorização, transação e suporte a acesso associativo. Gerencia também áreas associadas com sessões.

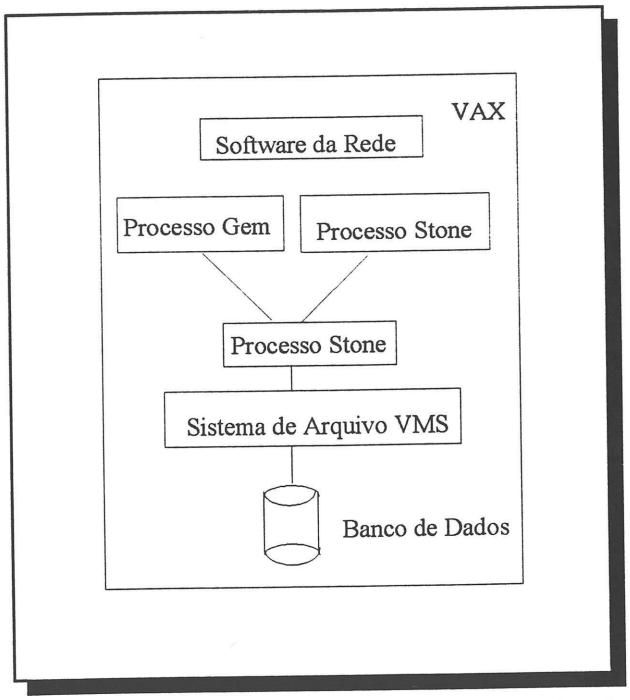

Figura 2.4 - Módulos de *Interface* Procedural

2.3.3 Versões do Modelo GemStone

Na **primeira versão** do GemStone foi utilizado um sistema de arquitetura distribuída, permitindo a manutenção de mais de seis réplicas de banco de dados na rede. A habilidade de distribuir cópias pela rede, auxilia na recuperação de falha de processador, permitindo que a máquina sob a qual uma cópia reside, assuma a funcionalidade que existia no processador com problemas de falhas. Essa habilidade permite também, que o processamento de dados recomece no evento que a máquina sob a qual o banco de dados residia deveria ser inacessível da rede [Butt91].

O controle de concorrência no GemStone pode ser realizado tanto por métodos otimistas como pessimistas. No esquema pessimista, uma implementação de transação tradicional é usada. No esquema otimista, uma cópia de segurança do espaço de trabalho do usuário é recebida no início de uma transação. Quando o cliente solicita o fim de uma transação, é feita uma verificação de conflitos com outras transações que tenham sido entregues desde o início da transação. Se algum dado lido ou impresso

pela transação for modificado por qualquer outra transação, a cópia de segurança é descartada e o GemStone informa ao cliente que a transação foi abortada [Catt94]. As páginas de dados originais não são removidas até que todas as transações que as tenham lido, tenham sido finalizadas ou abortadas. Desta forma, todo Gem tem uma visão compatível do banco de dados durante toda transação [Butt91].

Em resumo, o uso de controle de concorrência fornece três vantagens principais:

- **o uso do controle otimista** não requer o fechamento de páginas ou objetos, e *deadlocks* não ocorrem;
- **o uso de *shadowing*** fornece a cada transação, uma visão compatível do banco de dados: atualizações realizadas por uma transação não estão disponíveis a outras transações, até o fechamento da transação, e atualizações realizadas por outras transações não são visíveis até que a transação esteja encerrada ou tenha sido abortada;
- **o uso de *shadowing*** também fornece proteção contra falha em processo, processador e meios de comunicação [Butt91].

Na **versão posterior**, em virtude das limitações de tamanho de banco de dados GemStone existentes na versão anterior, foi feita uma combinação de vários tamanhos; coleta de lixo *on-line* e *backup*; e cópias permitiram que o GemStone fosse usado nas aplicações que necessitavam de quantidades volumosas de dados.

O gerenciamento do *cache* foi alterado nesta versão, visando uma melhoria no desempenho. Anteriormente, os objetos recuperados do banco de dados eram incluídos duplamente nos *caches*: uma vez na página *cache* e outra no objeto *cache*. Agora, os objetos recuperados do banco de dados são incluídos somente na página *cache*. O objeto *cache* é usado para objetos criados pela transação [Butt91].

Em seguida são descritos os tipos de *interface* suportadas pelo modelo GemStone.

2.3.4 *Interfaces*

a) A *interface* C

Essa *interface* é uma biblioteca de funções em C, que fornece uma ligação entre um código C de aplicação e o banco de dados GemStone, permitindo o acesso ao banco de dados, tanto estruturalmente (modelo C) quanto por envio de mensagens (o modelo DML- Data Manipulation Language -

GemStone). O servidor de objeto GemStone contém o esquema do banco de dados (definições de classe) e os objetos do banco de dados (instâncias das classes), embora o programa C possua as funções que envolvem controle e definição de *interface*-usuário.

A *interface* deve fornecer funções para as operações de acesso estruturais:

- transferir objeto de estrutura complexa - Uma apresentação do objeto fornece informação sobre a identidade do objeto, tamanho da classe, implementação e valores de variáveis de instâncias;
- tradução entre a máquina independente da representação da primitiva de dado usada no banco de dados, e máquinas dependentes da representação usada na aplicação;
- criar objetos novos no banco de dados;
- acessar e modificar o conteúdo interno dos objetos do banco de dados.

Além disso, as funções podem ser escritas em linguagem C ou C++ e incluídas na DML. Estas funções são chamadas ações do usuário, e se assemelham às primitivas disponíveis ao usuário em outros sistemas.

b) A *interface* C++

Essa *interface* fornece armazenamento persistente para aplicações C++ e possibilita o acesso a objetos persistentes armazenados no GemStone por aplicações escritas em outras linguagens. Os objetos armazenados no GemStone têm identidade, existem independentemente do programa que os criou, e podem ser usados por outras aplicações do banco de dados, incluindo aquelas escritas em outras linguagens.

A *interface* é implementada como um pré-processador baseado na sintaxe do padrão C++, e é produzida tanto por chamada de procedimento remoto como por versões *linkáveis*. Além disso, é fornecida uma biblioteca de classe, dando ao programador um conjunto padrão de definições para estruturas de dados usadas comumente, bem como as funções de manipulação e gerenciamento dos dados com o código C++ [Butt91].

Em seguida é descrito em detalhes o Modelo de Representação de Objetos.

2.4 Modelo de Representação de Objetos

O MRO será introduzido com suas características principais. Além da representação gráfica utilizada pelo MRO, é mostrada uma visão do esquema e do banco de dados, ou seja, como são representados os objetos. Além disso, é feita uma descrição da *interface* e o suporte a versões.

2.4.1 Introdução

O MRO foi elaborado no Departamento de Ciências de Computação e Estatística do ICMSC-USP em São Carlos, com o objetivo de permitir a construção de sistemas de gerenciamento do banco de dados que atendam às necessidades de manipulação de dados e representação de informações dos sistemas de apoio a projetos tradicionais de engenharia CAD, ou de ambientes de suporte ao desenvolvimento de sistemas CASE [Trai92].

O desenvolvimento de um SGBDOO denominado GEO iniciou-se em meados de 1988, com a incorporação gradativa dos conceitos do MRO. Atualmente um dos objetivos desse sistema é a validação prática e o estudo de como os conceitos do MRO e de Sistemas Orientado a Objetos em geral podem ser implantados em ferramentas reais.

2.4.2 Representação Gráfica do MRO

A especificação do MRO inclui três diagramas gráficos que permitem a representação de modelagens feitas no MRO. São eles:

- DRO - Diagrama de Representação de Objetos, que representa graficamente o esquema do banco de dados, através da modelagem dos tipos dos elementos envolvidos na aplicação;
- DRI - Diagrama de Representação de Instâncias, que permite a representação das instâncias dos tipos modelados no esquema, ou seja, os dados armazenados no banco de dados. O objetivo desse diagrama é de servir como exemplo e, portanto, apenas as instâncias de interesse para o exemplo devem ser mostradas;
- DHC - Diagrama Hierárquico de Colônias, que pemite a representação gráfica de como os dados estão logicamente agrupados no banco de dados.

Dentre os conceitos do MRO, temos:

a) Objetos

O MRO modela o mundo real representando eventos e conceitos através de **objetos**. Um objeto centraliza todas as referências e informações que são mantidas no sistema. Um objeto possui um identificador principal e pode possuir outros identificadores que são caracterizados como **sinônimos**. Além dos identificadores, um objeto é também caracterizado por um **tipo**, o que o classifica como possuindo determinadas características em comum com os outros objetos do mesmo tipo. O tipo de um objeto é modelado no esquema e representado graficamente utilizando um DRO, enquanto suas instâncias são parte do banco de dados e mostradas em um DRI, como pode ser visto na Figura 2.5.

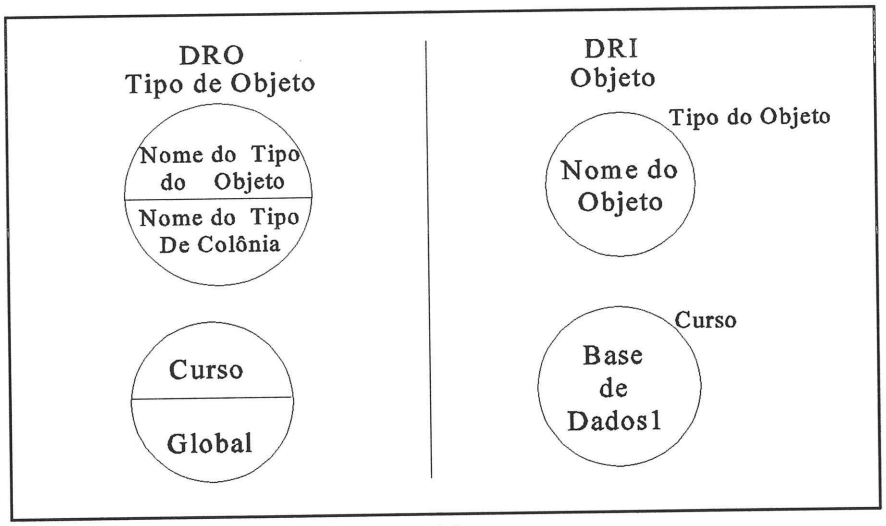

Figura 2.5 - Representação de Objetos

b) Relacionamentos

As associações entre objetos são definidas através de relacionamentos. São suportadas duas formas de relacionamentos:

- relacionamento binário - os quais sempre associam dois objetos, denominados **objeto origem** e **objeto destino**. Todo relacionamento segue um padrão definido no esquema denominado **tipo de**

relacionamento binário. Em todo relacionamento há uma modalidade de relacionamento, que caracteriza como os objetos se associam. Todo tipo de relacionamento tem um tipo oposto, e assim todo relacionamento também tem uma modalidade oposta, em que os objetos origem e destino trocam de posição, como demonstrado na Figura 2.6.

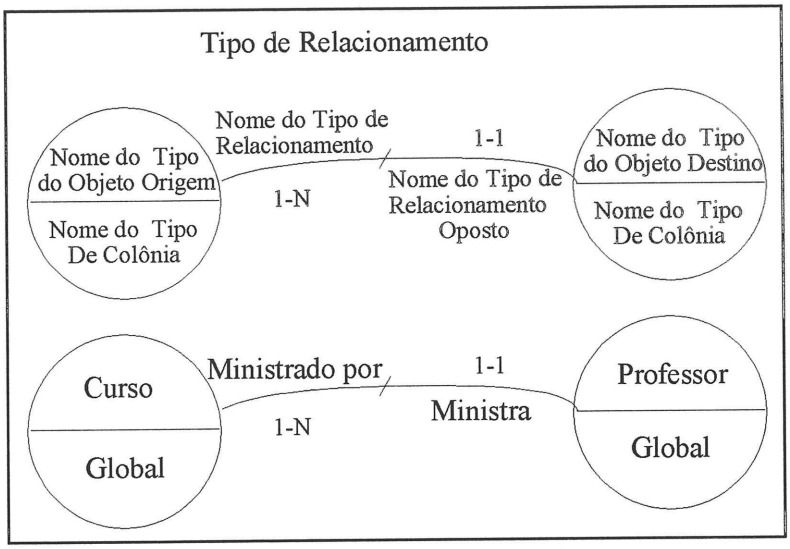

Figura 2.6- Tipo de Relacionamento Binário

- relacionamento triplo - neste relacionamento os objetos envolvidos podem ocupar três lugares distintos. Todo relacionamento triplo segue um padrão definido no esquema denominado **tipo de relacionamento triplo**.

c) Atributos

Assim como os objetos, os relacionamentos possuem **atributos**. Todos eles possuem um tipo e um conjunto de valores. A cada tipo de atributo está associada uma característica e o tipo de dado que ele contém, como mostrado na Figura 2.7. Um tipo de dado pode ser um inteiro, real, *byte*, cadeia de caracteres, ou outros. A característica de um atributo define o conjunto de operações que um gerenciador de dados pode efetuar sobre

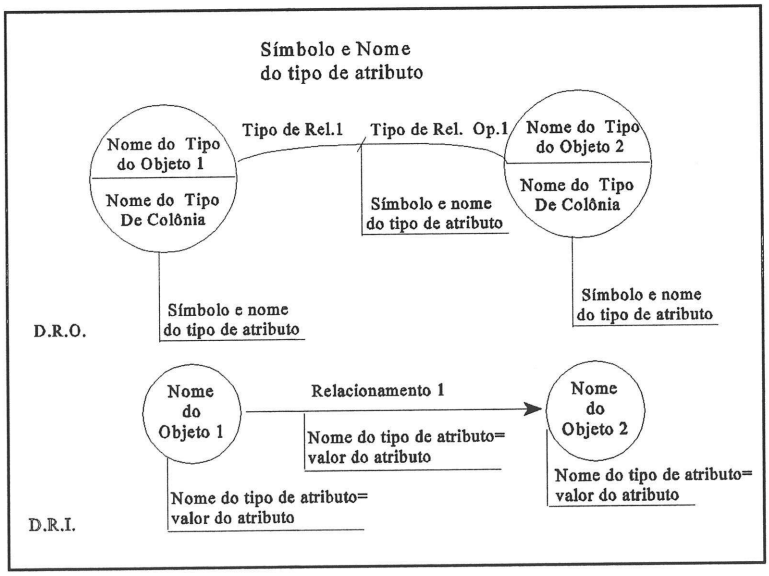

Figura 2.7- Tipo de Atributo

esse atributo [Biaj92].

Entre as características de um atributo, segundo [Biaj92], temos:

- **sinônimo** - identificador alternativo de um objeto;
- **comentário** - são informações para o usuário, e que não são analisadas pelo sistema;
- **regra** - atributos que possuem uma condição e uma ação;
- **procedimento** - são fórmulas ou parâmetros atribuídos a atividades que devem ser exercidas de uma forma pré-determinada pelo sistema;
- **tempo** - indicação de tempo, através de unidades de tempo ou indicações de data;
- **estrutura de dados** - agrupamento de informações com significado completo, que podem ser representadas em estruturas de dados como vetores e matrizes, ou para as quais possam ser definidas operações próprias de conjuntos, tais como união e intersecção;
- **gráfico** - corresponde a informação de localização no espaço, podendo indicar um ponto, uma função, um conjunto de segmentos de retas, entre outros;
- **imagem** - estes atributos representam figuras definidas através de valores associados a coordenadas cartesianas;
- **som** - são os sons digitalizados armazenados no banco de dados;
- **visualização** - atributos que definem como um elemento de um modelo pode ser visualizado em uma *interface* de entrada e saída do sistema, para permitir a interação do usuário com o aplicativo que manipule o objeto ou relacionamento ao qual está associado.

Em resumo, os elementos conceituais fundamentais do MRO são os objetos, classificados segundo seus tipos; relacionamentos, classificados de acordo com suas modalidades, as quais determinam os tipos de relacionamentos possíveis que envolvem tipos específicos de objetos; e atributos, também classificados pelo tipo e pela característica.

Além disso, cada conjunto de tipos de objetos pode formar uma colônia, sendo cada objeto pertencente à colônia denominado **habitante**. Uma das colônias está sempre disponível, e é chamada **colônia global**. Nessa colônia todos os objetos estão permanentemente disponíveis [Trai92]. Os tipos de colônias organizam-se segundo uma hierarquia, em que a colônia global é o topo da hierarquia. O nível imediatamente subordinado é constituído pelas colônias constritas por objetos de tipos que habitam

a colônia global. Essa hierarquia é formada por um relacionamento intrínseco entre o objeto que constringe a colônia e os objetos que a habitam [Biaj92].

Os tipos de objetos e tipos de relacionamentos podem ser especializados em **subtipos de objetos** e **subtipos de relacionamentos**. Um **subtipo de objeto** indica um refinamento de um tipo de objeto ou de um outro subtipo de objeto. Um **subtipo de relacionamento** indica um refinamento de um tipo de relacionamento ou de um subtipo de relacionamento. Por outro lado, alguns objetos podem ser semanticamente agrupados com outros e coletivamente se caracterizarem como supertipos.

Assim, subtipos representam um mecanismo para a representação de relacionamentos de generalização, ao passo que supertipos correspondem à existência de relacionamentos de sumarização entre objetos [Biaj92]. Em seguida são descritas características referentes à *interface* do MRO.

2.4.3 *Interface*

As características de atributos do grupo de *interface* correspondem a procedimento, gráfico, imagem e visualização. Essas características correspondem a tipos de atributos que permitem efetuar o acoplamento do Sistema de Gerenciamento do Banco de Dados com outros módulos gerenciadores que dão suporte a um aplicativo. Os **procedimentos** armazenam processos que devem ser repassados para um ambiente de suporte em tempo de execução ou ao sistema operacional. Os **gráficos** armazenam informações gráficas que podem ser repassadas diretamente a um núcleo gráfico, para a representação de figuras geométricas. As **imagens** são atributos cujos valores devem ser processados e/ou apresentados por sistemas de processamento de sinais, para a representação e manipulação de valores seqüenciais (voz) ou de figuras por varredura. As **visualizações** são tipos de atributos que armazenam uma forma de *interfaceamento* com o usuário, através de um Sistema de Gerenciamento de *Interface* com Usuários. Tipos de atributos com características deste grupo têm existência no banco de dados por si mesmos, porém, em geral, se referem a outros atributos dos objetos ou relacionamentos a que estão associados, e não são diretamente suportados pelo gerenciador do banco de dados, mas pelos demais módulos do sistema computacional.O suporte de versões e alternativas é descrito a seguir.

2.4.4 Versões e Alternativas

O conceito de colônias permite também a armazenagem de versões e alternativas de partes de um projeto, permitindo a geração de instâncias de colônias de objetos. Uma **colônia** pode ter várias instâncias, sendo cada instância chamada de uma **variante**. Podem existir variantes que permitem alterações no conteúdo da colônia, as quais correspondem a alternativas de projeto ainda em desenvolvimento; ou variantes já congeladas, denominadas **versões**, as quais não podem mais ser alteradas. Uma visão conceitual do MRO é mostrada e descrita em seguida.

2.4.5 Meta-Esquema de Gerenciamento de Dados

O **esquema de dados** de uma aplicação é um caso particular de uma colônia, visto na Figura 2.8, cujo conteúdo define a estrutura de dados de outras colônias. Assim, um **tipo de objeto** é apenas um objeto que habita o esquema que rege as colônias onde os objetos habitam. O tipo de objeto é um meta-tipo de objeto, reconhecido pelo sistema como o meta-esquema (a modelagem do MRO feita no próprio MRO), mostrado na Figura 2.8, a qual é reconhecida implicitamente pelo gerenciador e será explicada em detalhes no Capítulo 4. Um **esquema** é definido pelo meta-esquema, e as demais informações armazenadas em um banco de dados são esquematizadas pela colônia esquema [Biaj92].

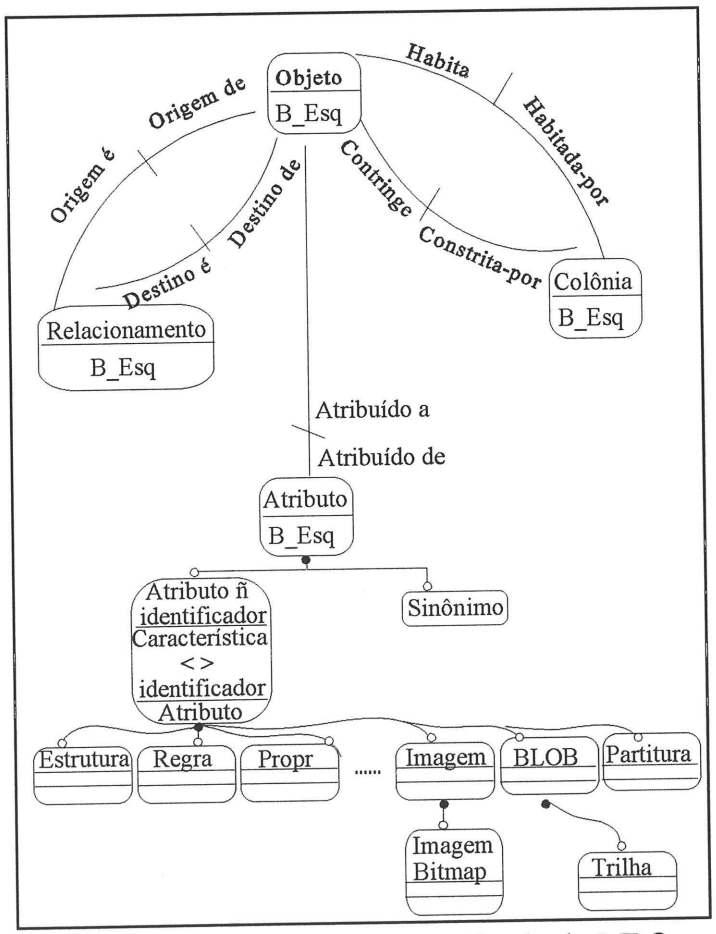

Figura 2.8 - Meta-Esquema Simplificado do MRO

2.5 Comparação dos Modelos de Dados Orientado a Objetos

Com base no estudo feito, foi possível verificar as características oferecidas por cada modelo, como pode ser visto no Quadro 2.1.

Quadro 2.1- Comparação dos Modelos de Dados Orientado a Objetos

Modelos	Características	Aplicação	Arquitetura	Suporte ao Tratamento de Áudio
O_2	-gerenciamento de objetos complexos; -reusabilidade de aplicação; -gerenciamento de rede; -aderência a padrões;	-CASE; -CAD/CAM; -multimídia; -gerenciamento de documentação	-criação, acesso, modificação e remoção de classes; -controle de consistência; -gerenciamento de objetos complexos; -gerenciamento de transação; -gerenciamento de disco; -controle de concorrência;	-armazena áudio como BLOB e possui ferramentas para edição de imagens e áudio;

Quadro 2.1- Comparação dos Modelos de Dados Orientado a Objetos (1a.Continuação)

Modelos	Características	Aplicação	Arquitetura	Suporte ao Tratamento de Áudio
ObjectStore	-gerenciamento de transação; -acesso distribuído; -acesso a dados persistentes; -modelagem de dados complexos; -facilidade de migração; -gerenciamento de versão; -reusabilidade de código;	-*groupware*; -gerenciamento de rede; -projeto de engenharia; -automação de processos;	-migração de código; -capacidade de modelagem de dados flexível;	-armazena áudio como BLOB;
GemStone	-suporte concorrente p/várias linguagens; -controle de transação multi-usuário; -segurança em nível de objeto;	-acesso associativo; -concorrência; -CASE; -CAD/CAM;	-controle de concorrência; -autorização; -transações;	-suporte para o BLOB;

Quadro 2.1- Comparação dos Modelos de Dados Orientado a Objetos (Continuação)

Modelos	Características	Aplicação	Arquitetura	Suporte ao Tratamento de Áudio
MRO	-gerenciamento de objetos complexos; -gerenciamento de versão e alternativa; -gerenciamento de rede; -aderência a padrões;	-desenvolvimento de sistema CASE e de engenharia; -aplicações científicas;	-objetos complexos distribuídos; -semântica para composição de objetos; -baseado na álgebra relacional e teoria dos grafos;	-permite definição de características do tipo partitura.

2.6 Considerações Finais

Este capítulo teve como objetivo mostrar algumas características de quatro modelos de dados orientado a objetos, enfocando principalmente a arquitetura de cada um deles. Esse estudo permitiu a verificação do suporte e adequação de cada modelo ao tratamento de áudio. Essa verificação, por sua vez, também possibilitou a escolha de um dos modelos, no caso o MRO, para aplicação do objetivo dessa dissertação.

CAPÍTULO 3
Definições e Conceitos em Áudio

Neste capítulo é dada uma visão geral de alguns conceitos e formas de representações de áudio em computadores, entre eles alguns tipos específicos de formatos de arquivo.O formato de arquivo MIDI foi apresentado detalhadamente, dando uma visão de como é feito o armazenamento de partituras.

3.1 Duas Tecnologias da Eletro Acústica: Analógico x Digital

Os **equipamentos analógicos** são aqueles em que o sinal elétrico, responsável pela representação da informação sonora, é processado na sua forma original. Este processo é feito sem a transformação em informações digitais usando uma grandeza física contínua,mesmo que o controle do equipamento seja digital e/ou contenha *displays*, ou outros elementos típicos de sistemas digitais. Nos **equipamentos digitais** existe uma conversão análogica/digital e/ou digital/análogica da informação sonora (conversão AD, DA). A conversão AD consiste em medir o som em intervalos regulares, e representam digitalmente cada medida efetuada.

Atualmente, os equipamentos analógicos são ainda muito usados, não só pelo fato de ser impossível utilizar exclusivamente a tecnologia digital, mas também por terem características de custo que, em relação ao desempenho, os tornam perfeitamente utilizáveis.

Além disso, existem correntes de pensamento no campo da música que preferem a utilização de sistemas mais tradicionais, quer seja pela pureza do som, ou simplesmente para obtenção de um som diferente daquele que se faz com os equipamentos de última geração. Atualmente, coloca-se em um único circuito integrado o equivalente a milhares das antigas válvulas, ocupando o espaço de menos de 1% de apenas uma delas.

Se isto, por um lado integra e miniaturiza os sistemas, por outro exige conversões do som em sinais digitais, causando alterações de timbre e conseqüentemente insatisfação dos músicos, que ainda optam pelo uso de válvulas. É impossível reproduzir, com um conjunto de duas a cinco caixas acústicas, o campo sonoro original. Embora a reprodução digital seja mais precisa do ponto de vista técnico, dado que distorções ocorrem de qualquer maneira, músicos com ouvidos apurados, usualmente preferem o "amaciamento" do som produzido pelos equipamentos valvulados, do que a "dureza fria" da reprodução, em particular amplificação, totalmente digital.

A seguir são apresentados alguns conceitos e características principais do som.

3.2 Som: Conceitos e Características Principais

De acordo com a acústica, que é uma parte da física, o **som** é o efeito produzido no aparelho auditivo pelas vibrações das moléculas de um meio transmissor, visto na Figura 3.1. Este meio transmissor normalmente é o ar, mas também existem outros meios possíveis [Gome93].

Para que se perceba o som, são necessários cinco elementos básicos:

- um corpo sonoro (elástico);
- um excitador das vibrações (dedo);
- um meio transmissor (ar);
- um orgão auditivo (ouvido);
- um ressonador, que repassa as vibrações para o meio transmissor ar (corpo do instrumento).

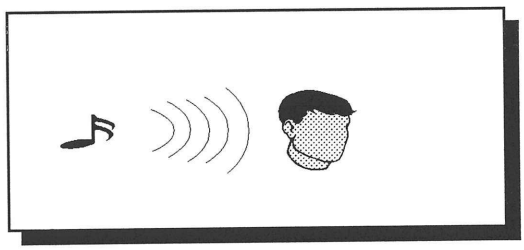

Figura 3.1 - Propagação do Som

Um exemplo da aplicação é o uso de um elástico entre dois dedos, mantendo este esticado, e quando batido com a outra mão produz um som. Neste caso, o corpo sonoro é o elástico, que se refere a qualquer elemento capaz de causar deslocamento nas moléculas do meio que o envolve. O dedo serve como excitador de vibrações, enquanto o meio transmissor é o ar, e o orgão auditivo é o ouvido [Gome93]. Nesse exemplo, o ressonador é o próprio elástico, que é muito pouco eficiente em transferência das vibrações para o ar. Se o elástico for encostado, por exemplo, em um ponto, o som será transferido mais facilmente, produzindo um campo sonoro mais intenso.

3.2.1 Características

As características referentes ao som são: volume ou intensidade, altura ou afinação, timbre ou conteúdo harmônico. Além disso, cada característica tem um componente estático e outro dinâmico, denominado **envelope**. A seguir são descritas com mais detalhes essas características.

a. Volume ou Intensidade

É a quantidade de energia aplicada ao corpo sonoro, para que ele produza vibração. Em áudio, a unidade usada para medir a intensidade sonora é conhecida como decibel - dB NIS-Níveis de Intensidade Sonora, como visto no Quadro 3.1. Fisicamente corresponde à amplitude da vibração.

Quadro 3.1 - Níveis de Intensidade Sonora

0 dB	Limite mínimo de audição
10 dB	Conversação em voz baixa
30 dB	Rua sem tráfego
70 dB	Tráfego intenso em grandes cidades
120 dB	Turbina de avião
130 dB	Limite máximo a partir do qual começa a surgir a sensação de dor

b. Altura ou Afinação

É a quantidade de movimentos executados pelo corpo gerador em uma determinada unidade de tempo. Normalmente arbitrada em um segundo, a unidade de medida é o Hertz-Hz (ciclos por segundo).

É possível verificar um movimento completo do corpo, quando há deslocamento do ponto 1 ao ponto 2, e daí para o ponto 3, passando pelo ponto 1 e retornando a origem, como visto na Figura 3.2.

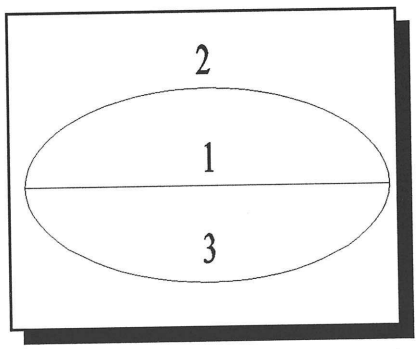

Figura 3.2 - Movimento de Oscilação de um Corpo

A seqüência completa, conhecida como um **ciclo**, somente encerra-se quando o padrão começa novamente a repetir-se. O ouvido humano tem capacidade de perceber frequências entre 20Hz e 20000Hz (20Khz).

As vibrações de frequência baixa são consideradas sons graves, enquanto as de frequência alta são os sons agudos. Para se determinar a frequência é adotado no campo dos equipamentos de áudio, que os sons graves vão até aproximadamente 300Hz, os médios estão entre 300Hz e 2khz, e os agudos acima de 2 Khz.

Na realidade, a característica altura do som classifica dois tipos de sons: os periódicos e os aperiódicos. Os **periódicos** são os que efetivamente apresentam um padrão de repetição periódica, e portanto possuem uma frequência. Os **aperiódicos** não possuem tal padrão, e portanto (embora tenham um conteúdo harmônico), não têm uma frequência definida. Exemplos de sons aperiódicos são os chiados e assopros.

c. Timbre

É a característica que depende única e exclusivamente da natureza do corpo sonoro. Fisicamente, corresponde à forma de onda. Na definição física do timbre, encontramos dois conceitos, o de frequência fundamental e o de harmônico. A **frequência fundamental** é a frequência própria da oscilação, desconsiderando as frequências que possam existir nos materiais dos instrumentos, enquanto os **harmônicos** são suas frequências múltiplas do fundamental. A forma de onda mais simples da natureza é a senóide. Segundo Fourier, qualquer forma de onda periódica pode ser descrita por:

$t = \text{sen}(f) + a_2 \text{ sen } (2f) + a_3 \text{ sen } (3f) + ...$

onde

f é a frequência fundamental;

a_2, a_3, ... são as amplitudes das senóides de frequência 2x, 3x,... a fundamental; por serem frequências com múltiplas intensidades da fundamental, são chamados de harmonias.

Assim sendo, o **timbre** pode também ser definido como o conteúdo harmônico do som gerado, que é determinado diretamente pelo material empregado na construção do corpo sonoro.

d. Características Dinâmicas

Tanto o volume, quanto a altura e o timbre podem ser (e usualmente são) dinâmicos. Isso significa que durante a emissão do som; essas características alternam-se constantemente, com a própria variação em si descrevendo uma curva (nesse caso em geral não repetitiva). Essa curva recebe o nome de **envelope**, ou **envolvente**.

O **envelope de volume** define como o volume de uma nota varia durante a emissão daquela nota. É muito comum esse envelope ser composto por quatro fases, usualmente descritas de maneira linear ou logarítmica:

❶ **Ataque** - corresponde ao surto inicial de energia, que dispara o som. Pode ser rápido, como em um som percursivo, em que o ataque corresponde ao primeiro "empurrão" dado no corpo sonoro, ou mais lento, como em um som de flauta, em que vários ciclos vão intensificando-se até atingir o volume "cheio".

❷ **Decaimento** - corresponde a uma primeira diminuição de volume depois do ataque, quando o som ajusta-se para ser depois mantido estável. Por exemplo, um som de violino tem um ataque de velocidade média, que atinge grande intensidade inicial, causando um grande "brilho" no início da nota, para diminuir rapidamente até um nível mais "suave", que pode ser sustentado por um período relativamente largo. A fase de decaimento corresponde à volta desde o pico do ataque, até o nível de sustentação.

❸ **Sustentação** - corresponde ao período da nota em que o volume permanece mais estável. Nem todos os instrumentos possuem este período tal como os instrumentos de corda percutidas.

❹ **Relaxação** - corresponde a uma diminuição de volume até sua extensão.

A curva correspondente a esse comportamento é chamada **ADSR**-Ataque, Decaimento, Sustentação e Relaxação. Essa curva varia de instrumento para instrumento (por exemplo, pode-se oscilar periodicamente o período de sustentação, causando o efeito conhecido como trêmolo), e é muito importante para auxiliar a identificação de um som.

O **envelope de altura** é bem menos significativo do que o de volume, pois variar a frequência de uma nota corresponde a "correr" a nota - um efeito produzido por exemplo na guitarra havaiana - e isso normalmente é percebido pelo dedo humano como uma tendência a desafinar. Pequenas alterações de frequência no início e final de uma nota, no entanto dão o colorido específico de cada instrumento e de seu executor. Variação periódica da sustentação em frequência corresponde ao efeito conhecido como vibrato.

O **envelope de timbre** é na realidade uma coleção de curvas, cada uma associada a uma harmônica, em geral do tipo ADSR, com os tempos de ataque e relaxação em geral sincronizados. Sons naturais costumam ter as curvas de envelopes harmônicos bastante complexas, e são na realidade a grande dificuldade para a síntese artificial de sons realísticos.

e. Ritmo

O **ritmo**, no contexto da música, é definido por Morehead [Ming95] como sendo o conjunto característico de sons e pausas no tempo, baseado na duração das notas, pressões fortes e fracas e outros fatores.

A seguir veremos que a geração do som pode ser feita de diversas maneiras.

3.2.2 Geração

Nos **instrumentos acústicos**, a geração do som é feita por meio do movimento físico de partes mecânicas. No caso de instrumentos de cordas, por exemplo o violão, a geração de som é dada pelo resultado da vibração da corda. Em um teclado semelhante ao piano, a tecla aciona um martelo que percute uma corda produzindo assim a vibração, que é amplificada pela caixa de ressonância.

Os **instrumentos elétricos** são aqueles nos quais os elementos mecânicos são utilizados para gerar o tom, mas este só pode ser claramente ouvido depois que for eletricamente amplificado. Um exemplo é a guitarra, onde o captador transforma a vibração mecânica da corda em impulsos elétricos, que são tratados pelo amplificador e pelas caixas acústicas.

Os **instrumentos eletrônicos** são circuitos eletrônicos, não havendo necessidade da existência de partes fisicamente móveis para a geração de som. Exemplos deste tipo são: orgãos, pianos eletrônicos sintetizadores e baterias eletrônicas [Gome93].

Assim como a geração, a gravação do som pode ser feita de várias maneiras. Estas são descritas a seguir.

3.2.3 Gravação do Som

A gravação do som pode ser feita, por exemplo, através de um microfone. Quando o sinal do microfone for incluído em um gravador de fita magnética, seu circuito transformará o sinal elétrico em sinal magnético, que fica registrado na fita. A partir dessa gravação é possível a reprodução do sinal magnético, bastando fazer a operação inversa, e tendo como saída para o som, um amplificador e alto-falantes.

Alguns processos de gravação em fita magnética apresentam algumas desvantagens em função do meio usado para gravação, ou seja, a fita magnética. Isso ocorre pois, quando o sinal elétrico é registrado na fita há uma perda de fidelidade, com a distorção do sinal original e presença de ruídos gerados na própria fita.

Porém, existem atualmente processos de gravação mais aprimorados, que usam tecnologia digital. Para converter um sinal elétrico em sinal digital, é necessário convertê-lo da forma analógica para forma digital, isto é, códigos numéricos que podem ser interpretados por microprocessadores.

A conversão de um sinal analógico para digital é feita por um circuito específico, denominado **conversor AD**. O processo corresponde a medir o sinal em intervalos de tempo pré-determinado, num processo chamado **digitalização de amostragem**, mostrado na Figura 3.3.

Cada valor medido é então "quantizado", isto é, o valor da medida não é qualquer, mas deve ser escolhido como o que mais se aproxima do real, dentro de um conjunto discreto pré-determinado. Esse processo é necessário pois o valor medido é representado num formato binário. Assim, por exemplo, se for usado um *byte* para representar o sinal, o valor deve ser escolhido entre apenas 2^8=256 alternativas possíveis. Quanto mais bits forem utilizados para representar cada amostra, maior a precisão,

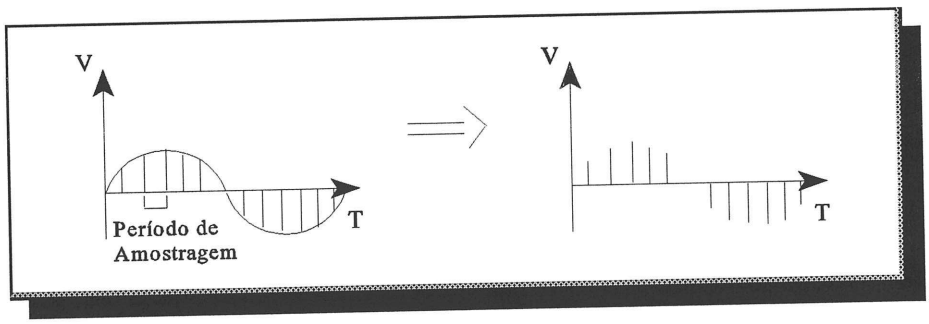

Figura 3.3- Período de Amostragem

e portanto melhor a fidelidade do som digitalizado. Na prática, utilizam-se 8,12,16,18,20,22 ou 24 bits por amostragem. Os discos de áudio CD -Compact Disc utilizam 16 bits, o que em geral é considerado suficiente para a armazenagem do som. Os equipamentos profissionais para processamento de áudio, tais como equalizadores e reverberadores, chegam a usar até 24 bits.

A frequência de amostragem também é fundamental para determinar a fidelidade do som. É possível demonstrar que a frequência máxima de um sinal a ser amostrado é igual à metade da frequência de amostragem (Teoria de Nyquist). Assim, quanto maior a frequência de amostragem, maior o número de harmônicos de um sinal que será registrado, e conseqüentemente maior sua fidelidade. O ouvido humando consegue ouvir sons até o limite de 20 Khz. Assim, registrar-se frequências maiores do que essa, não melhora o som que pode ser ouvido. A qualidade de um CD áudio, amostrado em 44.1 Khz é considerada ótima, pois permite gerar sons de até 22.05 Khz. Profissionalmente utilizam-se taxas de até 48 Khz, para permitir uma filtragem mais intensa do sinal amostrado.

Note-se que quanto maior a frequência de amostragem, maior a necessidade de memória para armazenar o som digitalizado. Por exemplo, para o padrão de um CD de áudio (padrão *red-book*), amostram-se dois canais (stéreo) em 44.1 Khz e 16 bits por amostra. Isso implica em 2*44.100x2 *bytes*/segundo = 176,4 K*bytes*/segundo, ou 10,584 M*bytes*/minuto de gravação.

Na reprodução do áudio digitalizado, emprega-se um circuito conversor digital/analógico (conversor D/A). Nesse processo, cada valor amostrado é sustentado pelo período dessa amostra, recriando aproximadamente a forma de onda original, como mostrado na Figura 3.4.

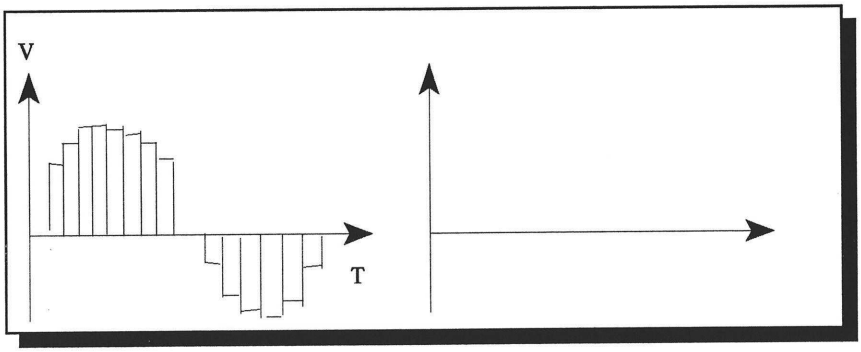

Figura 3.4 - Processo de Recriação da Forma de Onda Original

Todo erro da diferença entre a curva original e a curva recriada, é devido a dois processos empregados na digitalização:

❶ **Quantização** - esse erro somente pode ser minimizado pelo aumento na precisão do valor amostrado;

❷ **Amostragem** - esse erro pode ser muito diminuído aplicando uma filtragem no sinal recriado, através de um filtro passa-baixa, que desconsidera o ruído abaixo de uma certa frequência, com frequência de corte igual à metade da frequência de amostragem. Como todo ruído de amostragem tem uma frequência no mínimo igual à frequência de amostragem, esse ruído é muito atenuado, e não sobrecarrega os estágios seguintes de processamento do som (agem analógico somente) sem causar perda alguma no espectro audível.

Na Seção 3.3 serão descritas as principais formas de representação de áudio em computadores.

3.3 Formas de Representação de Áudio em Computadores

Existem basicamente duas maneiras de representar áudio, uma é representando um som digitalizado a partir de uma gravação de um sinal originado fora do computador - usualmente através de microfones, ou outros captadores de sons do mundo real - essa maneira é denominada **representação de áudio digitalizado.**

A segunda maneira consiste em representar um conjunto de parâmetros que permitem recriar o som, sintetizando-o a partir desses parâmetros. Essa maneira é denominada **representação de áudio por síntese.**

É possível ainda armazenar o som de uma maneira híbrida.

Sobre o **áudio representado**, pode-se aplicar diversos operadores, que transformam o som original. O conjunto de operadores aplicáveis é diferente, dependendo da maneira como o som é representado.

Sobre **áudio digitalizado** pode-se aplicar operações tais como filtragem, equalização, eco, reverberação e manipulação do envelope de volume global. Não é possível o tratamento de cada componente individual do som e tal como um dos instrumentos gravados, a menos que exista uma gravação independente para cada componente.

Sobre **áudio representado sinteticamente**, pode-se aplicar operações tais como controle das operações individuais de envelope (volume, timbre e frequência) de cada componente, alterações no andamento da música sem alterações na altura, mudança em cada instrumento, etc.

Em ambas as formas de representação, diferentes tecnologias podem ser empregadas. Por exemplo, sons digitalizados podem ser amostrados por uma tabela de quantização logarítmica, e incremental, podem ser comprimidos, e podem ter os tempos de amostragem variáveis ou interpolados. A representação de um som digitalizado também varia, embora a mesma tecnologia seja empregada.

As tecnologias de síntese de som mais empregadas atualmente são duas:

❶Modulação em Frequência FM

❷ Reprodução de amostras - Amostragem

Ambas podem ser empregadas de inúmeras maneiras, embora a tecnologia de FM-Frequence Modulate tenha sido amplamente difundida pelos circuitos integrados OPL2 e OPL3 da Yamaha, através das placas de som *Sound Blaster* da *Creative Labs*, tendo tornado-se um padrão de fato. No âmbito dos equipamentos PC, o formato FM é sinônimo da representação de áudio seguindo a especificação desses dois integrados (dos quais o OPL3 é um superconjunto do OPL2). No caso da tecnologia *sampler*, não existe por enquanto um formato que se sobressaia.

3.3.1 Formatos de Arquivos

Os arquivos de som e música são identificados por suas extensões de arquivo e por alguns *bytes* de sentinela inseridos na estrutura do arquivo. Estas extensões representam o padrão necessário para tocar a música eletrônica. Existem vários formatos de arquivos padrões, entre eles:

a. MIDI

O termo MIDI significa Interface Digital de Instrumentos Musicais - Musical Instrument Digital Interface. Este formato permite a representação de músicas, não através de sons diretamente, mas representando-se as notas que devem soar nos instantes adequados. Ele permite a transmissão de informação musical entre instrumentos eletrônicos e computadores. Embora criado para permitir a troca de sinais entre instrumentos musicais, foi posteriormente desenvolvido um padrão para a armazenagem

desses sinais, junto a temporização correspondente, em arquivos de computadores. O MIDI não permite o tratamento com voz humana, entretanto o MIDI apresenta duas vantagens importantes, com relação ao seu tamanho, pois não ocupa muito espaço, e com relação a qualidade que é muito boa [Zhan95].Um arquivo MIDI tem extensão .mid. Na Seção 3.5 este padrão será explicado em detalhes.

b. MOD - MODulation

O formato *soundtracker* é o formato de arquivo mais usado nos sistemas Amiga. Os módulos *soundtracker* são encontrados em jogos, demonstrativos (demos) e outros tipos de programas.

O conceito básico do formato ***soundtracker*** é usar modelos de instrumentos digitalizados e proporcionar a estes modelos o tom desejado, alterando a frequência do *playback* [Stol93]. Os modelos são chamados **instrumentos**, embora eles não tenham que ser necessariamente modelos de instrumentos reais. O uso de um som de violino, um instrumento oriental exótico ou uma voz humana, pode ser usado como um modelo [Zhan95]. Isto possibilita a criação de partes musicais realistas, pois os instrumentos modelados podem ser mais realistas do que os sintetizados [Stol93].Além disso, como cada instrumento deve ser chamado na memória somente uma vez, o resultado são arquivos menores do que partes de músicas completamente digitalizadas. Logo, é possível a criação de músicas longas sem muita informação. É permitido ainda, alterar o som de um instrumento de diversas formas, como por exemplo, adicionando vibrato. O formato original da *soundtracker* pode manipular 15 instrumentos digitais diferentes e 4 simultaneamente, podendo ainda ser expandido para até 31 instrumentos e 16 simultaneamente [Stol93].Um arquivo MOD tem extensão .MOD [Zhan95].

O formato **MOD** é um híbrido entre a representação do som digitalizado (armazenam digitalmente os modelos de instrumentos) e a representação de som sintetizado (quando representa as notas e os instantes em que elas devem soar), de uma maneira muito parecida à empregada em arquivos MIDI. Sendo um híbrido, o formato MOD apresenta muitas vantagens e características de ambas as representações, embora demande bastante do *hardware* disponível.

c. WAV

O formato WAV Wave Riff é atualmente o mais comum para a armazenagem de sons digitalizados. Nesse formato, a seqüência de valores amostrados são armazenadas em disco, um valor

de cada canal amostrado de cada vez, numa taxa de amostragem constante indicada em um cabeçalho do arquivo.

A Microsoft desenvolveu seu próprio formato, descrevendo estruturas de arquivo Riff-Resource Interchange File Format. Um arquivo Riff é formado por componentes chamados *chunks*. Um ***chunk*** é um segmento de dado lógico que tem sempre a mesma estrutura, independente do tipo de dado existente [Stol93]. Nos arquivos Wave, os *chunks* de dados são sons digitalizados, representação de um sinal analógico, que são diferentes dos arquivos MIDI que não contêm dados de sons, mas listas de comandos para dispositivos MIDI. Portanto, os arquivos Wave são arquivos de dados enormes, que ocupam grande espaço em disco para tocar música, por somente um minuto. Um arquivo Wave tem extensão .Wav [Zhan95].

d. SBI e IBK

Quando utiliza-se uma representação de um som sintetizado, como o formato .Mid por exemplo, é necessário dispor de um *hardware* específico, que constitui o sintetizador propriamente dito. Teclados, módulos de expansão e placas de som contém sintetizadores **multi-timbrais**, ou seja, podem gerar sons de diversos instrumentos simultaneamente. Cada som é individualmente gerado por um circuito denominado **gerador de tom - *tone generator***. Para que um gerador de tom sintetize um timbre de um determinado instrumento, este deve ser programado (parametrizado) adequadamente.

Existem muitas tecnologias para implementar-se sintetizadores, e cada uma delas é parametrizada de maneira específica - um programa feito para um sintetizador é incompatível com qualquer outro sintetizador.

Devido à grande disseminação de placas de som, utilizando sintetizadores de FM implementados nos circuitos integrados OPL2 e OPL3 da Yamaha, utilizados por exemplo nas placas *Sound Blaster da Creative Labs*, os programas de timbre para esse sintetizador tem um formato de arquivo padronizado,com a extensão .SBI - *Sound Blaster Instrument*. O formato SBI descreve como um instrumento pode ser sintetizado a partir de um conjunto de parâmetros que controlam um sintetizador que utiliza a técnica de geração de sons por modulação em freqüência. O formato SBK-Sound BanK permite o armazenamento de vários instrumentos representados no formato SBI em um único arquivo.Um arquivo .IBK - *Instrument Bank* armazena até 128 programas para instrumentos, num

formato muito semelhante ao formato .SBI.

e. SoNG

O formato SNG-SoNG é equivalente ao formato MOD, porém representa os instrumentos através da parametrização para sintetizadores operando por modulação em freqüência.

3.4 *Interfaces*

O computador possui *interfaces* que permitem a comunicação da unidade central de processamento com o ambiente exterior. Existem *interfaces* especificamente projetadas para aplicações em MIDI.

Os dispositivos de *interface* podem ser encontrados em duas versões: como placa de circuito, conectada diretamente no computador, ou como equipamento adicional, que utiliza uma linha de comunicação padrão disponível na maioria dos computadores.

Os tipos de *interface* a serem instaladas são:

1. *Interfaces* MIDI

Este tipo de *interface* permite ao computador a comunicação com os dispositivos MIDI, através de *software* específico. Estas *interfaces* são classificadas de acordo com o número de portas e/ou canais disponíveis. Todas suportam 1, 2 ou 4 portas, sendo as mais complexas com 8 portas, e 16 canais MIDI por porta [Gome93].

Basicamente há dois tipos de *interfaces* MIDI, as **internas** e as **externas**.

A ***interface* interna** como mostrada na Figura 3.5, é uma placa eletrônica inserida em um dos conectores internos de expansão, por onde o computador entrega e acessa os dados na *interface*. Na face externa da placa estão os conectores MIDI, que permitem interligá-la aos instrumentos musicais.

Na ***interface* externa**, o circuito eletrônico

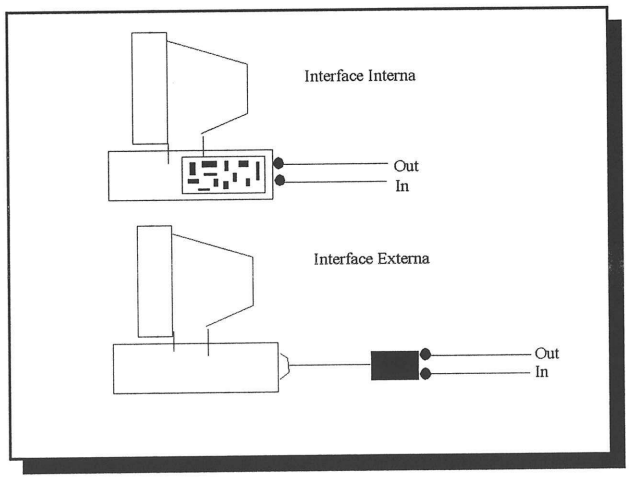

Figura 3.5 - Tipos de *Interface* MIDI

está em uma caixa pequena, que é ligada ao computador através da porta paralela ou através da porta serial.

As *interfaces* internas são mais baratas pois dispensam acabamento, enquanto as externas são a solução para usuários de computadores portáteis, que não podem usar placas internas.

Assim como qualquer outro dispositivo, a *interface* MIDI necessita ser identificada dentro do computador. Para isso, é usado um endereço, código pelo qual a *interface* é acessada pelo processador central, e um número de interrupção [Ratt95].

2. *Interfaces* para Áudio Digital

São conversores digital/analógico e analógico/digital. Como **conversor analógico para digital** este permite que o computador grave sinais de áudio, tanto de voz como de instrumentos. Como **conversor digital para analógico**, permite que o computador reproduza seus próprios timbres, independente de onde estes estejam gravados.

3. *Interfaces* Multimídia

Essas *interfaces* permitem a manipulação de equipamentos de áudio e vídeo. Numa placa para multimídia, incluem-se as funções de uma porta MIDI, um sintetizador para áudio dispondo de diversos canais e dois canais de gravação e reprodução de áudio digital, tudo isso condensado em uma placa [Gome93].

Para garantir um *hardware* mínimo para os desenvolvedores de aplicações multimídia, foi criado um padrão para o que pode ser um computador pessoal para multimídia - Multimedia Personal Computer - MPC. O padrão original (nível 1) foi suplantado pelo atual nível 2. Essa especificação é mostrada a seguir no Quadro 3.2.

Quadro 3.2 - Quadro Padrão para Computador Pessoal Multimídia

	Nível 1	Nível 2
Processador	386SX-16MHz	486 - 25 MHz
Memória RAM	2 Mbytes	4 Mbytes
Disco	30 Mbytes	160 Mbytes
Monitor Vídeo	VGA 640x480-16cores	VGA 640x480-true color
Aúdio Digital	8 bits-22KHz-mono	16 bits-44KHz-estéreo
Midi	In-out	In-out-thru
Aúdio Sintetizado	6 carris+percursão	9 carris+percursão
CD-ROM (Taxa de Transferência)	150 Kbytes/seg	300Kbytes/seg-padrão XA E Kodak multisessão

3.5 Sistemas MIDI

A *Interface* Digital para Instrumentos Musicais é um protocolo de comunicações digitais, ou seja, é uma linguagem de controle padronizado e especificação de *hardware*, que permite a comunicação em tempo real entre instrumentos musicais e outros dispositivos [Hube95].

No início dos anos 70, os sintetizadores eram instrumentos muito populares no mundo da música [Gome93]. Era comum que dois sintetizadores fossem conectados juntamente, para que ambos instrumentos pudessem ser manipulados através do teclado de um deles [Pent90]. Porém, apresentavam uma grande desvantagem, eram **monofônicos**, ou seja, emitiam uma nota de cada vez, e não permitiam que os músicos trocassem de timbre em apresentações ao vivo.

Houveram várias tentativas para solucionar o problema, até o surgimento de equipamentos polifônicos. Apesar disso trazer algumas vantagens, não foi feita uma padronização para o uso dos timbres. Isso permitiu que cada fabricante utilizasse seus próprios timbres de acordo com sua adequação, gerando insatisfação, já que foi necessária a sobreposição de timbres [Gome93]. A tecnologia que permitiu a construção de sintetizadores polifônicos exige o emprego de um microprocessador embutido no instrumento.

Finalmente, na década de 80, foi desenvolvido um padrão que permitia entre outras coisas, a "conversa" entre os equipamentos. Este padrão foi chamado de MIDI [Gome93] surgindo de uma

padronização aceita pelos principais fabricantes de instrumentos musicais eletrônicos, de como utilizar uma *interface* serial para a troca de informações entre os processadores dos sintetizadores polifônicos.

O MIDI não foi utilizado apenas como sucessor de métodos anteriores, ele foi escolhido em razão da enorme aceitação. Ele foi resultado de discussões envolvendo vários fabricantes de instrumentos eletrônicos, incluindo Roland, SCI e Yamaha. Existem dois aspectos do MIDI - o *hardware* e o *software*. O ***hardware*** é o circuito eletrônico que passa informação de um instrumento para outro, e o ***software*** é o sistema de código usado para colocar quais as instruções complexas existentes e quais as que são realmente sinais elétricos [Pent90].

Com a sofisticação dos instrumentos atuais, a vantagem do MIDI foi que este proporcionou um meio simples de obter interconexão complexa entre instrumentos, enquanto os métodos antigos necessitavam de um grande número de cabos usados para a conexão [Pent90]. Apesar disso, o MIDI não é a *interface* perfeita para geração de som em algumas aplicações. Com a evolução dos equipamentos para geração de som, MIDI está se tornando um pouco limitado. Em razão do MIDI ter sido desenvolvido antes do surgimento de algumas técnicas de síntese, é necessário em determinadas circunstâncias, usar extensões à especificação original, bem como usar algumas instruções específicas do equipamento. Assim é possível um aproveitamento dos melhores aspectos, de geração de som, daquele equipamento em particular [Ming95]. O próprio formato de arquivos MIDI (na realidade três formatos diferentes) é uma extensão à especificação original.

3.5.1 Pontos Básicos

O MIDI foi definido originalmente como sendo um protocolo de comunicação, que estabelece regras básicas para que dois ou mais equipamentos possam trocar informações e comandos, e que pode ser usado por qualquer equipamento que adote uma comunicação do tipo serial.

Para tornar essa comunicação prática, foram padronizados três conectores,

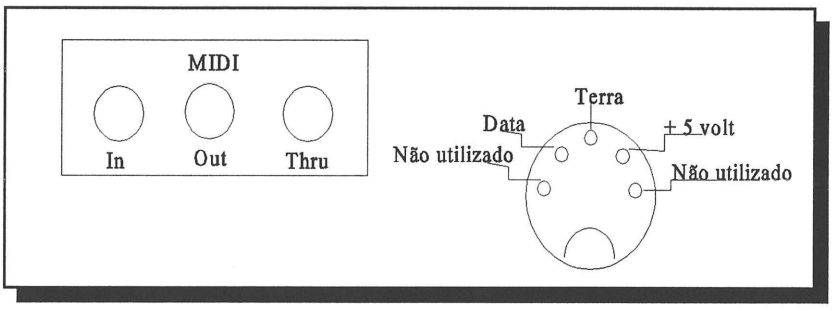

Figura 3.6 - Conectores DIN Usados para MIDI

todos do tipo DIN de cinco pinos chamados *in*, *out* e *thru*, mostrados na Figura 3.6:

A ***porta* MIDI *in*** recebe mensagens de uma fonte externa e passa a execução, controle e temporização de dados para o microprocessador interno do dispositivo. A **porta MIDI *out*** é usada para transmitir mensagens de uma única fonte para o microprocessador de outro instrumento ou dispositivo MIDI. A **porta MIDI *thru*** propaga uma cópia exata dos dados incluídos na porta MIDI *in*.

Um ponto importante, é que através dos cabos MIDI não passam os sons propriamente ditos mas apenas informações digitais, responsáveis pelo controle das quatro características básicas do som a ser produzido, pelos equipamentos que receberam estes sinais, além de vários outros parâmetros que facilitam a comunicação entre vários equipamentos.

3.5.2 Mensagens MIDI

Através de mensagens é que se faz a transmissão dos dados pelo sistema MIDI. Elas podem ser divididas em dois grupos: **mensagens de canal e mensagens do sistema**.

a. Mensagens de Canal

São mensagens relacionadas à execução da nota, à forma pela qual cada uma é executada e ao modo de operação do equipamento. Elas podem ser divididas em dois grupos, como mostrada nas Figuras 3.7 e 3.8:

1. Mensagens de Voz - são as mensagens responsáveis pela transmissão das notas que soam (e que param de soar) em cada canal.

2. Mensagens de Modo - estas mensagens determinam como os sintetizadores, ou outros equipamentos escravos, receberão as mensagens enviadas pelo controlador. Uma diferença desta para as mensagens de voz, é que para estas nem todos os equipamentos MIDI são capazes de responder a todas as mensagens. Na verdade, as mensagens de modo são consideradas alterações de controle.

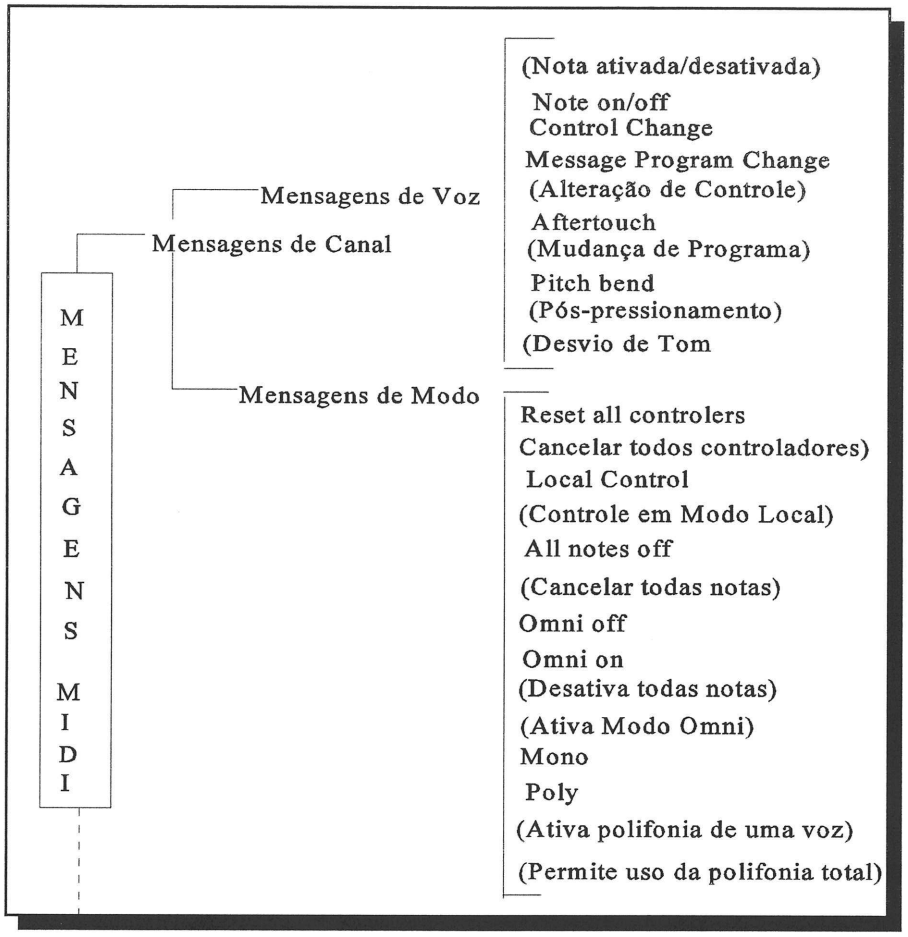

Figura 3.7 - Mensagens MIDI

b. Mensagens de Sistema

As **mensagens de sistema** diferem das mensagens de canal, pois elas não afetam diretamente a forma pela qual o som é produzido [Gome93]. Elas são transmitidas para todo dispositivo MIDI existente na cadeia. Isto significa que qualquer dispositivo responderá a estas mensagens, independente dos canais a que o dispositivo estiver atendendo [Hube95].

Elas são divididas em três grupos:

1. Mensagens Comuns do Sistema - permitem o controle de vários equipamentos que estejam executando a mesma música [Gome93].

3.5.3 Configurações Típicas

Existem várias permutações possíveis com o equipamento MIDI, permitindo que dispositivos sejam facilmente conectados a qualquer sistema MIDI. Estas configurações comuns, permitem que dados MIDI sejam distribuídos da forma mais eficiente possível.

Existem dois métodos de conexão entre dispositivos MIDI, como mostrado na Figura 3.10:

1. conectar a porta MIDI *out* de um dispositivo, na porta MIDI *in* de outro dispositivo;
2. conectar a porta MIDI *thru* de um dispositivo, na porta MIDI *in* de outro dispositivo.

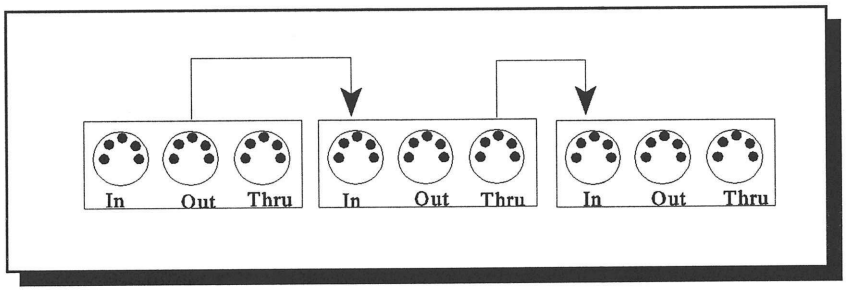

Figura 3.10 - Meios de Conexão de um Dispositivo MIDI

O Daisy Chain

Este é um dos métodos mais simples e mais comumente usados para distribuição de dados em um sistema MIDI. Este método é usado para distribuir uma única linha de dados MIDI para todos os dispositivos presentes no sistema, através da transmissão de dados para o primeiro dispositivo, e subseqüentemente passando uma cópia exata destes dados, através da porta MIDI *out* para o dispositivo seguinte na cadeia, como mostrado na Figura 3.11. Este processo é contínuo até que o dispositivo final seja alcançado [Hube95].

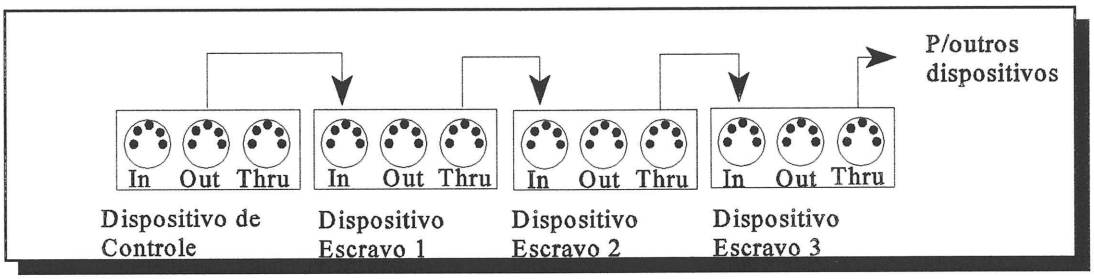

Figura 3.11 - Exemplo de um Sistema MIDI Conectado Usando uma *Daisy Chain*

Como a porta MIDI *thru* somente copia os dados da porta MIDI *in*, o sinal pode ser repassado em cada dispositivo a partir de um único controlador *master* (tal como um *sequencer* por exemplo). Em muitos casos, isto é aceitável pois o controlador é usado para transmitir dados para um ou mais canais MIDI, que serão respondidos um por vez, pelos dispositivos que tenham sido designados para estes canais. Qualquer dispositivo, pode ser usado como uma fonte controladora simplesmente *plugando* sua porta MIDI *out* à porta MIDI *in* do primeiro dispositivo da cadeia MIDI [Hube95].

A rede estrela

Este tipo de interconexão permite que um controlador *master* se comunique com uma quantidade de instrumentos MIDI sobre portas MIDI endereçáveis individualmente, como mostrado na Figura 3.12 e 3.13. A rota de dados com tal rede permite que cada conjunto de portas MIDI *in* e *out* acomodem fluxos isolados independentemente dos dados MIDI. Isto fornece maior flexibilidade nas interconexões do sistema.

Em sistemas MIDI mais complexos, uma rede estrela oferece muitas vantagens em relação a rede *daisy chain*.

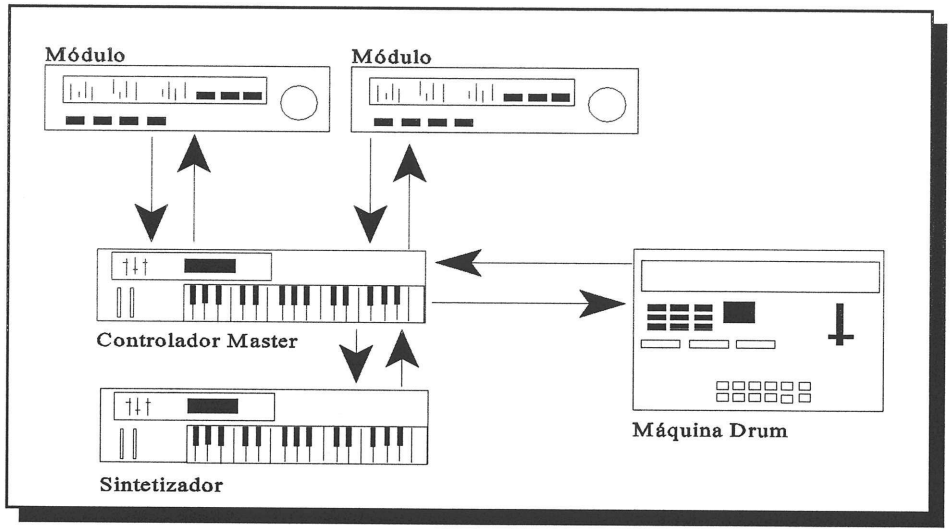

Figura 3.12 - Sistema de Interconexão Básico

Ele permite que um controlador direcione cada divisão de uma rede, para comunicar dados (via portas MIDI *in* e *out*) que é relevante a cada instrumento ou cadeia de instrumentos. Como os 16 canais dos dados MIDI isolado podem ser transmitidos e recebidos em cada divisão, este tipo de rede é ideal também para uso em conjuntos de dispositivos MIDI que precisem manipular mais que 16 canais [Hube95].

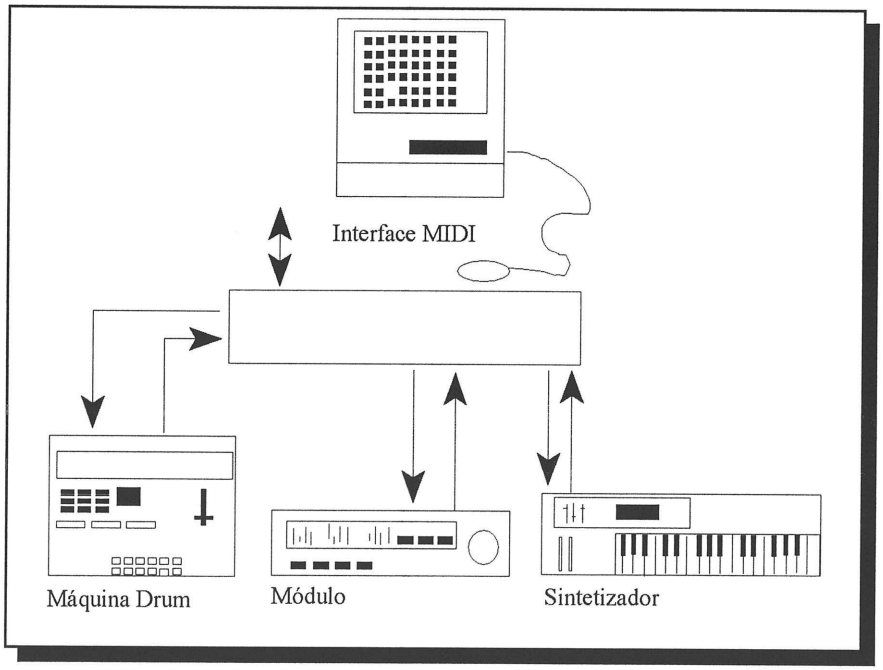

Figura 3.13 - Uso comum em Gerações Recentes de *Interface* MIDI

Outro aspecto interessante é que, por ser uma comunicação do tipo serial, apenas uma mensagem pode ser enviada ou recebida de cada vez.

Isto significa que mesmo que estejamos ouvindo duas notas juntas, na verdade os inícios de suas execuções estão separados por um minúsculo intervalo de tempo, assim como seus términos.

A taxa de transmissão do padrão MIDI, ou seja, a velocidade na qual a comunicação é feita, é de 31250 bits por segundo que, pelo fato da comunicação ser do tipo serial, são transmitidos um de cada vez. As mensagens MIDI são normalmente compostas por 3 *bytes* para informação, mais um bit para sinalizar o início e outro o término de cada *byte*. Assim uma mensagem MIDI normalmente é composta por 30 bits, o que resulta numa velocidade de mais de 1000 mensagens por segundo, rápido o suficiente para música [Gome93]. Isso significa também que as notas de um acorde começam e param de soar com um intervalo mínimo de 1/1000 segundo. Esse tempo é denominado **resolução do tempo (ou andamento)**, medido em termos musicais como **batidas por minuto**. A resolução MIDI é assim de 60 Kbpm. Para efeito de comparação, os melhores *softwares* disponíveis apresentam resolução de 576 Bpm.

3.5.4 Linguagem MIDI

As mensagens MIDI são compostas por palavras de 8 bits, que são transmitidas na fase serial, para converter um conjunto de instruções para um ou mais dispositivos MIDI em um sistema.

Existem dois tipos de *bytes* que são definidos na especificação MIDI: o ***byte status*** e o ***byte de dados***. Os ***bytes status*** são usados na mensagem MIDI como um identificador, para instruir o dispositivo receptor de que a função e o canal MIDI estejam sendo direcionados.

O ***byte de dados*** é usado para codificar o valor numérico atual que é colocado juntamente com o *byte de status* [Hube95].

Os canais MIDI permitem que as mensagens sejam enviadas para um dispositivo ou ainda para uma coleção de dispositivos, no caso de termos diversos equipamentos interligados em cascata. Isso é feito pelo *nibble* (quatro bits) no *byte de status*, que informam a todos os dispositivos receptores qual mensagem está sendo transmitida e por qual canal. Como o canal *nibble* está acima de 4 bits, mais que 16 canais podem ser transmitidos através de um único cabo.

Entretanto um dispositivo MIDI responde a apenas um número específico do canal, ignorando todas as mensagens do canal que são transmitidas para outro canal. Assim, qualquer dispositivo que for setado para responder a um canal MIDI específico, responderá somente as mensagens que são transmitidas para aquele canal [Hube95].

A programação dos canais MIDI pode ser feita através de duas configurações, programando

diversos equipamentos para responder no mesmo canal MIDI, permitindo assim a sobreposição de timbres, ou utilizando sintetizadores multitimbrais, possibilitando a execução de diversos timbres ao mesmo tempo [Gome93].

O *bit* mais significativo de um *byte* de status é 1, logo o *byte* será sempre 1xxxxx; enquanto que para o *byte* de dados é 0 estando sempre 0xxxxxx. Um *byte* de status é enviado primeiro, seguido de um ou dois *bytes* de dados, dependendo da mensagem.

Para tocar uma nota em um canal, duas mensagens são necessárias:

Note on - inicia a nota

Note off - finaliza a nota

sendo ambas mensagens de voz para canal.

O ***byte*** de status para *note on* é 1001xxxx, onde xxxx é o número binário do canal desejado. Então o *byte* de status 10010000 envia uma mensagem *note on* para canal 0 e 10011111 envia uma mensagem *note on* para canal 15.

O *byte* de dado seguinte, descreve a nota a ser tocada, na qual existem 128 valores possíveis. O segundo *byte* de dados de uma mensagem *Note On* contém os dados de velocidade. Uma nota para de tocar com a mensagem *Note Off*, sendo que o *byte* de status é 1000xxxx, onde xxxx é o número do canal. O primeiro *byte* de dado contém o número da nota e o segundo o parâmetro de relaxação do envelope ADSR [Stol93].

3.5.5 General MIDI

O padrão MIDI não especifica quais instrumentos devem soar em cada canal, embora a mensagem 1100xxxx especifique que o *program* indicado no *byte* de dados seguinte, deva ser atribuído ao canal xxxx. Um ***program*** é o conjunto de parâmetros que ajusta o sintetizador para reproduzir um som (e portanto um timbre, ou instrumento em particular). O formato MIDI permite assim a escolha de um entre 128 possíveis timbres para cada um dos 16 canais.

Um dos problemas relacionados à falta de padronização dos tipos de timbres, é que cada fabricante usava a numeração que achava mais conveniente. Um exemplo disto era quando usava-se um determinado equipamento, por exemplo X, que considerava o timbre 26 como sendo de um piano e o timbre 47 de um trompete; enquanto no equipamento Y o timbre 26 era classificado como violino e o 47 um contrabaixo. Uma música programada para tocar em X, quando tocada em Y soa no mínimo

estranha. Para solucionar este problema a *International* MIDI *Association* - IMA em 1991 criou uma padronização denominada **General MIDI -GM**, permitindo que qualquer seqüência musical criada de acordo com a especificação do padrão GM fosse executada corretamente em qualquer equipamento que atendesse o padrão GM. O padrão GM fez a padronização de 128 timbres, numerados de 1 a 128, além de outras características básicas.

A criação do padrão GM trouxe muitas vantagens para o uso de equipamentos musicais por leigos, permitindo que através da conexão de um módulo de som GM compatível com o seqüenciador, houvesse execução adequada das seqüências [Ratt95]. Este padrão no entanto não tem aplicação no mundo profissional, onde é necessária a liberdade para a criação ilimitada de novos timbres e nuances.

3.5.6 Tipos de Equipamentos

Dentre os dispositivos MIDI, temos os mostrados na Figura 3.14:

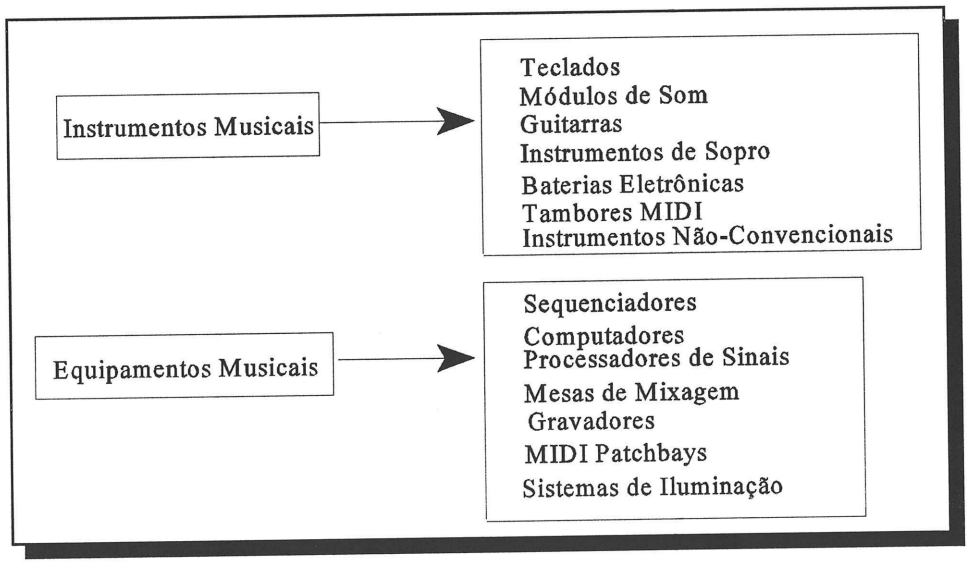

Figura 3.14 - Dispositivos MIDI

O ***sequencer*** (seqüenciador) é um equipamento que permite a gravação de mensagens MIDI, sincronizando-os no tempo, como mostrado na Figura 3.15.

Existem dois tipos de *sequencers*, o ***hardware sequencer*** e o ***software sequencer***, tendo como diferenças a forma construtiva e o computador empregado [Gome93].

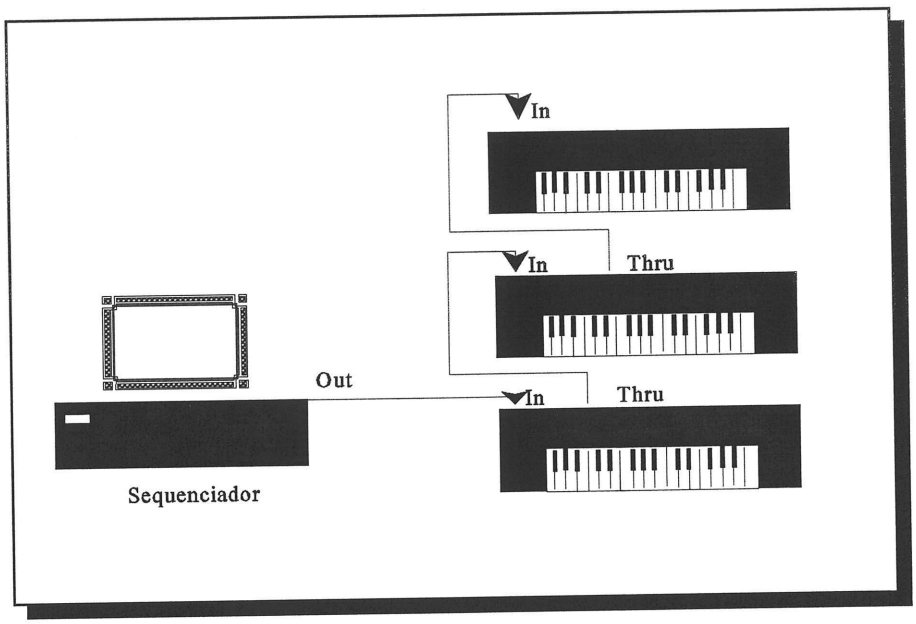

Figura 3.15 - Seqüenciador

O seqüenciador pode ser um equipamento portátil específico, ou um computador equipado com *interface* MIDI e rodando um *software* para seqüenciamento.

Dentre os **teclados** existem os pianos digitais, os sintetizadores, os *samplers*, que são sintetizadores onde o músico pode definir a origem do timbre, entre outros. Os **módulos de som** são os instrumentos que não possuem teclado, sendo projetados para serem usados como "escravos", controlados por seqüenciadores, ou outros controladores. Atualmente a maior parte dos equipamentos geradores de som incorporam mais de um instrumento, e que, via MIDI, funcionam como se fossem diversos instrumentos separados. Esses equipamentos são os chamados **multitimbrais**, como mostrado na Figura 3.16.

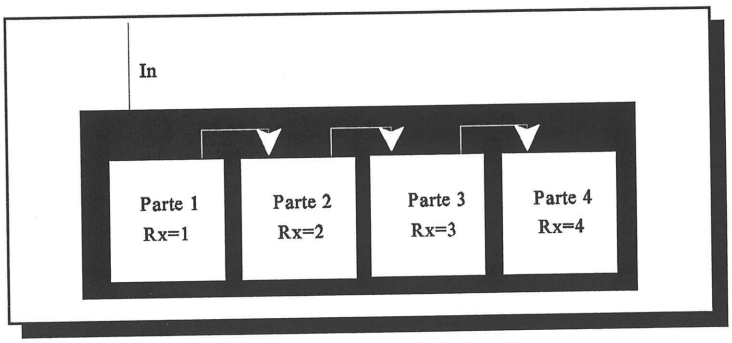

Figura 3.16 - Instrumento Multitimbral

As guitarras MIDI são guitarras elétricas que possuem um sistema captador/conversor capaz de detectar as notas da guitarra e traduzí-las nos comandos MIDI correspondentes. Os instrumentos de sopro possuem um funcionamento semelhante ao das guitarras. Dentre os **instrumentos não-convencionais**, temos o Theremin, que é um dispositivo com antenas, que capta os movimentos da mão do instrumentista e gera notas e controles MIDI.

Entre os **dispositivos auxiliares**, há o seqüenciador, que permite a composição de músicas, e será melhor explicado na Seção 3.5.7; o computador que passou a fazer parte desta categoria graças aos *softwares* para seqüenciamento, impressão de partituras, entre outros; e os processadores de sinais que são responsáveis pela produção de efeito sobre os sons dos instrumentos [Ratt95].

Os *sequencers* atuais são computadores que executam um programa de seqüenciamento MIDI [Gome93].

3.5.7 Os Seqüenciadores

Uma vez armazenada a seqüência de eventos e preservada a sua cronologia e um seqüenciador, é possível efetuar o procedimento inverso, retransmitindo ao instrumento todos os comandos, reexecutando-se exatamente o que fora executado anteriormente. Dessa forma, o músico pode gravar sua execução no seqüenciador, sob a forma de comandos MIDI; aqueles eventos, uma vez armazenados, podem ser reexecutados comandando o instrumento, atuando como "escravo", como mostra a Figura 3.17.

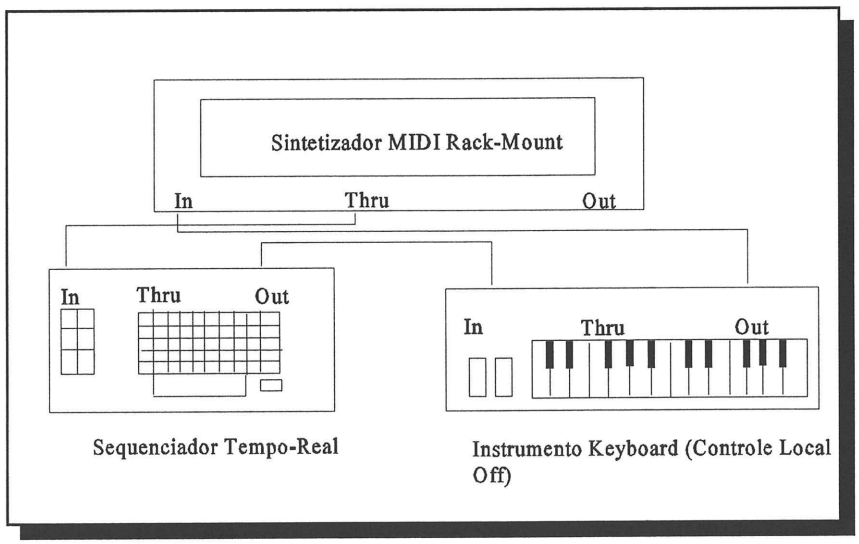

Figura 3.17 - *Setup* apropriado para Seqüência em Tempo Real

Um seqüenciador suporta múltiplas "pistas", permitindo-se construir uma música através de várias sessões de gravação. Assim um único músico, gravando um instrumento por vez pode gravar uma orquestra interna.

Outra grande vantagem do seqüenciador está nos recursos de edição, que permitem que o músico corrija trechos de gravação, sem ter que gravar todo o material novamente. Além disso, este pode realizar cortes e colagens de trechos, inserir eventos, etc.

3.5.8 Arquivos Padrão MIDI

Todos os seqüenciadores MIDI têm a capacidade de guardar seqüências em memória, no disco flexível ou em uma fita cassete. Porém, cada um deles usa métodos diferentes para guardar estes dados. Existe um certo grau de compatibilidade entre programas seqüenciadores de mesmo fabricante, mas em geral os dados de um seqüenciador não são lidos corretamente em outro seqüenciador. Para solucionar este problema foi criado o SMF - *Standard* MIDI *Files* - formato padrão para arquivos MIDI (extensão .mid) [Pent90], permitindo que vários programas possam permutar dados MIDI em tipos diferentes de computadores [Stol93].

Com o padrão SMF, cada *software* ou seqüenciador pode continuar possuindo seu próprio formato, admitindo mais recursos do que aqueles padronizados pelo SMF. Pelo SMF, porém podem reconhecer e gravar dados pelo padrão SMF, permitindo, por exemplo, a transferência de um seqüenciador para outro. Graças a esse padrão, é possível usar um determinado seqüenciador para criar a música, salvá-la em disco sob a forma de SMF e carregá-la no *software* editor de partituras para imprimir a partitura daquela música [Ratt95].

Entretanto, este formato foi desenvolvido muito depois da linguagem MIDI.

A estrutura dos arquivos SMF é baseada principalmente na idéia do padrão IFF -Interchange File Format [Stol93].

Os dados são divididos em blocos, que incluem um nome e um valor de 32 bits, seguido pela parte de dados de tamanho livre. É possível a expansão do formato, sem problemas de compatibilidade com versões novas ou antigas. Até agora somente dois blocos foram definidos e padronizados:

- **parte do *header*** - é identificado pela sentinela Mthd;
- **parte do *track*** - é identificado pela sentinela Mtrk.

Um **bloco *header*** deve aparecer no início de cada arquivo, seguido pelos *blocos tracks* [Stol93].

Foram definidos três tipos de arquivos SMF, como mostra a Figura 3.18:

• **arquivo MIDI de tipo 0** - neste arquivo toda seqüência é arquivada em apenas um bloco, que contém todos os dados seqüenciais da música [Ratt95]. Este arquivo é o tipo mais simples de todos [Stol93].

• **arquivo MIDI de tipo 1** - é muito versátil, pois permite várias trilhas por vez, sendo estas armazenadas sempre seqüencialmente dentro do arquivo. Entretanto, eventualmente estas trilhas contêm informações que devem ser interpretadas simultaneamente. Conseqüentemente, uma vez que todas as trilhas tenham sido chamadas, elas devem ser processadas cronologicamente paralelas em relação às outras [Stol93]. Usualmente cada trilha corresponde a apenas um canal MIDI e pode existir mais de uma trilha associada a um mesmo canal, embora esse vínculo, definido pelo aplicativo e pelo músico, não seja obrigatório.

• **arquivo MIDI de tipo 2** - embora este tipo também armazene nas trilhas seqüências diferentes, os dados contidos nestes tipos não são manipulados simultaneamente. Ao invés disso, eles podem ser divididos independentemente. Neste tipo de arquivo MIDI, é recomendável armazenar somente seqüências que pertençam a uma única parte da música, mesmo podendo armazenar qualquer coisa [Stol93]. Esse tipo é muito pouco utilizado.

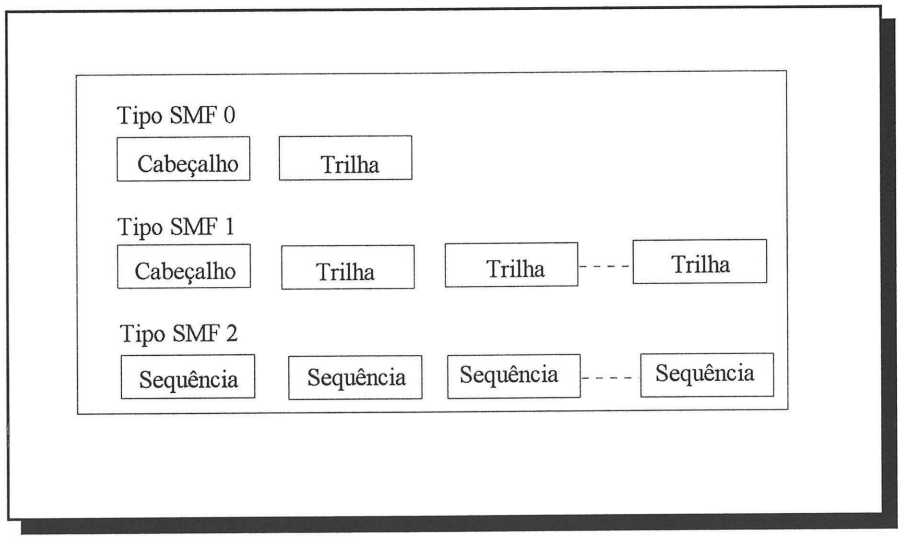

Figura 3.18 - Arquivos Padrão MIDI

O **bloco *header*** sempre é o primeiro em um arquivo SMF. É formado por um id (identificador) de 4 *bytes* com os caracteres Mthd e um valor de 4 *bytes*. O comprimento do *header* mais frequentemente usado é 6 *bytes*. Conseqüentemente, o comprimento dos *bytes* são seguidos por 6 *bytes* de dados. Os primeiros 2 *bytes* contêm o tipo do arquivo SMF, isto é, o valor 0,1 ou 2.

Quando há avaliação dos valores da palavra de um arquivo SMF, é necessário lembrar que a seqência de *bytes Lower* (baixo) e *Higher* (alto) são exatamente o oposto da seqüência usada pelos processadores Intel. Isto significa que deve ser feita a troca dos *bytes* altos e baixos antes de obter o valor correto.

Os próximos 2 *bytes* contêm o número de trilhas presentes nos arquivos SMF, como um valor da palavra. Para arquivos SMF de tipo 0, este valor é sempre 1.

Os dois *bytes* seguintes indicam o andamento da música, e devem ser interpretados diferentemente, dependendo do bit 15 ser ou não setado. Se o bit 15 não for setado, então os bits 14-0 representam um valor que indica a resolução com a que a música foi gravada, em colcheias por minuto. Por exemplo, os *bytes* 00000000 01100000 especificam 96 colcheias por minuto.

Entretanto, quando o bit 15 é setado para 1, isto indica que o código do tempo absoluto foi usado por este arquivo MIDI. O **Método MIDI do Código do Tempo-MMCT** é usado, por exemplo, para processamento de vídeo tape. Cada localização do vídeo tape é impressa com a informação do tempo. Assim, qualquer localização no tape pode ser acessada.

Uma vantagem deste processo é que o vídeo tape pode ser cortado e emendado em frações de segundo. Os intervalos de tempo usados, correspondem a *frames* por segundo que podem ser especificados em horas, minutos, segundos, *frames* e centésimos de *frames*.

Quando os vídeos de música são editados, tanto os dados do vídeo como o acompanhamento da música, devem ser precisamente combinados. Esta é uma das razões pelas quais os formatos de música usam esse método de código de tempo. Este método permite que o vídeo e os dados da música sejam indicados juntamente em frações de segundo [Stol93].

O MMCT é o método de codificar tempo absoluto preferido. Este método necessita somente de um quantidade mínima de memória, além dos dados MIDI.

Com relação ao método de código do tempo, há mais uma informação a respeito. Quando o bit 15 é setado, os bits 14-8 representam um valor na forma de um complemento de dois negativo, que indica a quantidade de *frames*, por segundo, que o código de tempo especifica. Os bits de 7 a 0 indicam a resolução do sinal do código de tempo.

Um bloco da trilha começa com o id (identificador- Mtrk) e os 32 bits referentes ao tamanho do bloco. Estes são seguidos por qualquer número de eventos, que formam os dados reais da trilha. Cada evento é precedido por um valor, que indica quanto tempo deve passar, antes do evento ser executado. Este valor se refere ao **delta de tempo** em relação ao evento anterior. Os ***bytes*** do delta de tempo são componentes de sintaxe permanente de um evento no formato de arquivo MIDI [Stol93].

Para economizar espaço, o valor do delta de tempo não tem um número fixo. Ao invés disso, ele usa o bit 7 do *byte* para indicar que virá outro *byte* e comprimento.

Conseqüentemente, somente os bits de 6 a 0 são usados atualmente para o tamanho do valor. Teoricamente, isto permite que qualquer número de *bytes* possam ser concatenados, formando números do tamanho desejado. Entretanto, deve ser usado no máximo quatro *bytes*. Isto significa que a menor seqüência de delta de tempo é um *byte* do tipo 0xxxxxxx, que terá valores entre 0 e 127. A seqüência mais longa do tempo delta é 1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx. Para representar o número 255 na notação delta, é necessário usar mais que um *byte*. Usando um segundo *byte*, este número deveria ser: 10000001 01111111.

A seqüência de delta de tempo é seguida pelo número do evento. Três tipos diferentes de eventos podem existir em um arquivo SMF. Estes são **eventos MIDI**, **mensagens exclusivas do sistema** e **meta eventos**.

Os **eventos MIDI** correspondem aos relacionados diretamente com a execução musical, como o ato de pressionar e soltar teclas, indicação de instrumentos, as variações de volume, uso de controladores, e demais parametrizações na geração do som. O Quadro 3.3 mostra os eventos MIDI existentes e seu significado, e os parâmetros associados a cada um.

Quadro 3.3- Eventos MIDI

Eventos MIDI	**Significado**	**Parâmetros**
Note On	Tocar uma nota	canal, nota e intensidade
Note Off	Terminar uma nota	canal, nota, velocidade de harmonia
Control Change	Alterar parâmetro	canal, número do parâmetro (1= modulador, 7= volume, etc) e valor do parâmetro
Program Change	Alterar instrumento	canal, novo instrumento associado a este canal
Polyphonic Aftertouch	Alterar nota que está tocando	canal, nota, valor do parâmetro
Aftertouch	Alterar parâmetro de canal	canal, valor do parâmetro

As **mensagens exclusivas do sistema** são eventos não padronizados, específicos de cada fabricante de instrumentos para permitir acomodar as particularidades de cada instrumento. Visto que os conteúdos destas mensagens não são determinados pelo padrão MIDI, somente deve ser incluído o tamanho da informação para esta mensagem. Um mensagem exclusiva do sistema em um arquivo SMF começa com o *byte* $F0. Este *byte* é seguido pelo comprimento do dado seguinte, ao invés do primeiro *byte* de dado. Este valor é codificado da mesma maneira que o valor de delta de tempo.

O *byte* final da mensagem exclusiva do sistema é o *byte* $F7. Como o tamanho do dado já foi especificado a rigor, este *byte* não é necessário. Entretanto, ele é guardado por outra razão.

Nos arquivos MIDI, é possível dividir tamanhos de mensagem exclusivas do sistema em pacotes gerenciáveis. Isto habilita o dado do código do tempo ser incluído em lugares desejados. A segunda parte de cada mensagem dividida começa com o *byte* $F7. Embora a mensagem não deva começar com $F0, esta deve sempre terminar com $F7, para indicar o fim da mensagem exclusiva do sistema.

Os **meta eventos** são específicos dos arquivos SMF. Eles podem conter vários tipos de dados, como mostrado no Quadro 3.4. A função de cada meta-evento é descrita abaixo.

Quadro 3.4- Meta-Eventos

Meta-evento MIDI	**Significado**	**Atributo pré-definido**	**Característica do Atributo**
0	Número seqüencial da trilha	Sequence-number	Propriedade Número Inteiro
1	Texto livre	Texto-da-Partitura	Comentário
2	Copyright	Patente	Comentário
3	Nome da Trilha	Descrição-da-Trilha	Comentário
4	Nome do Instrumento	Nome-do-Instrumento	Comentário
5	Letra da Música	Letra	Comentário
47	Fim da Trilha	-	-
81	Define Andamento	Andamento	Propriedade Número Inteiro
84	Offset para Código de Temporização MIDI (SMPTE)	Código-de-Tempo	Propriedade Número [5] *byte*
88	Resolução de Tempo	Resolução-de-Tempo	Propriedade Número [4] *byte*
127	Dados do Seqüenciador	Seqüenciador	Comentário

• **Evento $00 (00) - Número da Seqüência**

Se o evento 0 é incluído no arquivo SMF, este deve ser colocado no início da trilha, antes do primeiro dado MIDI que deve ser transferido. Este evento tem 2 *bytes* de dados representando um valor de 16 bits, contendo o número da seqüência dos próximos dados. Isto é importante principalmente para arquivos SMF do tipo 2. O evento pode ser usado para seqüências de "número" contidas em tal arquivo. Se estes eventos são omitidos, a mesma seqüência encontrada no arquivo é usada.

• **Evento $01 (01) - Texto Livre**

Este evento permite a inserção de qualquer texto no arquivo SMF. O *byte* do id (identificador) $01 é seguido pelo tamanho do texto. Como em outros eventos de texto, este é representado no formato delta. Este código é seguido pelo texto em formato ASCII.

• **Evento $02 (02) - Copyright**

Este evento também contém texto. Entretanto, o evento de número 2 é reservado especificamente para aviso de direitos autorais. Os avisos de direitos autorais são colocados na primeira trilha do arquivo.

• **Evento $03 (03) - Nome da Trilha**

O evento 3 fornece outra forma de armazenar texto no arquivo SMF. Novamente, o tamanho do valor é armazenado no formato delta. O número 3 é reservado para caracterizar a informação na trilha de dados ou informar sobre a seqüência de dados. É possível armazenar títulos em geral, tais como *bass intro* ou *drum solo*; além disso, é possível identificar os dados de música armazenados nesta trilha. Em arquivo SMF do tipo 2, é possível usar este evento para armazenar informação na seqüência.

• **Evento $04 (04) - Nome do Instrumento**

Neste evento é informado o som do instrumento. Novamente, o tamanho do valor é armazenado no formato delta. Uma aplicação é a indicação do instrumento apropriado a ser tocado no canal apropriado.

• **Evento $05 (05) - Letra de Música**

Quando estes eventos são utilizados, tanto a música como as letras são incluídas no arquivo SMF. Em geral o evento do texto de uma música pode ser inserido no começo de cada sílaba que será tocada.

• **Evento $06 (06) - Marcação**

O evento 6 pode ser usado para marcar pontos específicos em uma seqüência, já que estes podem ser localizados facilmente mais tarde. Novamente, o tamanho do valor é armazenado no formato delta.

• **Evento $07 (07) - Fila de Pontos**

Este evento é útil quando arquivos MIDI são usados com material de vídeo. Uma fila de ponto é um ponto para o qual você pode indicar voltar ou retroceder. Assim, o texto descritivo no evento 7 pode conter informação naquele ponto em particular.

• **Evento $08 -$0F (08-15) - Novo**

Os eventos de 8 a 15 são reservados para eventos de texto, embora eles não tenham sido especificados.

• **Evento $20 (32) - Prefixo do Canal**

Este evento é usado para mostrar que todos os eventos meta são direcionados a um canal específico. O número do canal é armazenado em um único *byte*, seguindo o evento id. Esta seleção de canal permanece ativa até que um novo evento MIDI seja encontrado.

Conseqüentemente, o evento 32 não desvia os dados MIDI para outro canal. Ao invés disso, ele é usado somente para finalidades de informação. Portanto, ele pode ser usado com o evento 4, desde que o número do canal de um instrumento não tenha sido especificado.

• **Evento $2F (47) - Fim de Trilha**

Este evento indica o fim de um bloco de trilha. Como não seguem dados adicionais, o tamanho deste evento é 0. Entretanto, devido a notação de eventos padronizada, este tamanho do valor é armazenado também no formato delta após o evento id. Um evento 47 deve ser sempre colocado no final da trilha.

• **Evento $51 (81) - Define Andamento**

Seguindo aos *bytes* de tamanho, o evento 51 contém um número de 24 bits. Este número representa o número de microsegundos em cada colcheia, fazendo com que um valor de 1.000.000 indique que uma colcheia deva ser tocada a cada segundo. Esse evento define o andamento da música, na resolução indicada pelo bloco *header*. Um evento $51 somente pode ser inserido no primeiro bloco de trilha do arquivo.

• **Evento $54 (84) - Offset para Código de Temporização MIDI**

Este evento é usado para parametrizar o método de código de tempo MMCT. Este evento contém 5 *bytes* de dados, que são os parâmetros para esse código.

• **Evento $58 (88) - Resolução de Tempo**

Neste evento há 4 *bytes* de dados além do valor do comprimento delta.

• **Evento $7F (127) - Dados do Seqüenciador**

O padrão SMF contém 127 eventos para acomodar os diversos seqüenciadores. Este evento é muito parecido com as mensagens exclusivas do sistema, pois eles podem ser facilmente adaptados a requisitos de *software* ou *hardware* por fabricantes do seqüenciador [Stol93].

O fato de salvar uma música no formato SMF não garante o reconhecimento por qualquer seqüenciador. Para que isso aconteça o seqüenciador deve ser capaz de "entender" o formato do próprio disquete. Devido a grande popularidade alcançada pelos computadores compatíveis com o padrão PC, tem havido uma forte tendência para que todos os seqüenciadores portáteis e mesmo computadores sejam capazes de ler e escrever em discos padrão PC.

A compatibilidade do formato de arquivamento de seqüências, aliado ao padrão de timbres definido no General MIDI, viabilizou a disseminação de músicas MIDI em disquete, distribuídas e comercializadas em todo o mundo. Atualmente, existem diversas empresas que se especializam em criar, sob a forma de seqüências SMF os sucessos internacionais de vários gêneros. Nessas seqüências, são usados somente comandos General MIDI, de forma que elas podem ser reproduzidas em qualquer equipamento compatível [Ratt95].

3.6 Considerações Finais

Este capítulo apresentou um levantamento geral da terminologia de áudio, que foi utilizada como base para uma terminologia uniforme utilizada nos capítulos seguintes.

O levantamento incluiu conceitos de áudio, desde as formas de gravação até as formas de representação, permitindo a escolha da forma usada neste trabalho, que no caso foi a síntese do som.

Dentro da representação, foram apresentados alguns formatos de arquivos, e mais especificamente o arquivo MIDI já que esse foi o escolhido para aplicação neste trabalho.

CAPÍTULO 4
Suporte de Áudio no MRO

Este capítulo mostra o tratamento e a representação definidos para a característica partitura no Modelo de Representação de Objetos. Além disso, são mostrados alguns comandos da Linguagem de Acesso ao MRO que foi estendida para a representação da partitura musical.

4.1 Descrição do Tipo de Dado Partitura como uma Característica

A representação de áudio, conforme visto no Capítulo 3, pode ser feita de duas maneiras: pelo áudio digitalizado, que considera que um som é digitalizado e representado a partir de uma gravação de um sinal originado fora do computador usualmente através de microfones, ou outros captadores de sons do mundo real; e através do áudio sintetizado o qual representa-se um conjunto de parâmetros permitindo recriar o som, sintetizando-o a partir desses parâmetros. Essa maneira é denominada representação de áudio por síntese. É possível ainda armazenar o som de uma maneira híbrida.

As técnicas de síntese são adequadas para a representação de sons musicais. O mesmo ocorre para sons ambientais ou voz, uma vez que a parametrização desses últimos é geralmente difícil, pois sua representação através de expressões matemáticas é muito complexa e não realista.

Em ambas as formas de representação, diferentes tecnologias podem ser empregadas. Por exemplo, como visto no Capítulo 3, sons digitalizados podem ser amostrados por uma tabela de quantização linear, logarítmica ou incremental, podem ser comprimidos, e ter os tempos de amostragem variáveis ou interpolados. A representação de um som digitalizado varia, mesmo quando uma mesma tecnologia é empregada. Em uma aplicação as tecnologias de síntese de som mais empregadas atualmente são duas: Freqüência Modulada - FM e reprodução de amostras , cuja técnica é chamada de Amostragem, como visto no Capítulo 3.

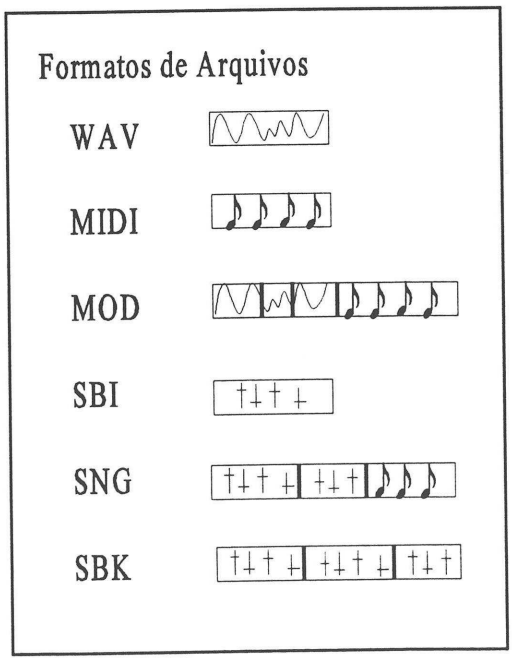

Figura 4.1- Representação de Sons

Dependendo da representação usada, existem vários padrões para armazenar sons em meios digitais, como

mostrado na Figura 4.1. O formato WAV - Wave Riff é uma técnica usada para o armazenamento de sons digitalizados. Nesse formato, a seqüência de valores amostrados são armazenados em disco, sendo que um valor de cada canal é amostrado por vez, numa taxa de amostragem constante indicada num cabeçalho do arquivo. O formato MIDI, descrito no Capítulo 3, atualmente é o mais utilizado para armazenagem de seqüências musicais, onde apenas as notas das músicas e sua temporização são armazenadas [Gome93], [Hube95], [Stol93], [Ratt95]. Esse formato não permite a representação de músicas através dos sons diretamente, mas representando-se as notas que cada instrumento deve soar nos instantes adequados. Devido ao tipo de dados armazenados, freqüentemente refere-se ao tipo de dados que um arquivo em formato MIDI armazena como uma "partitura" musical.

Outros formatos podem ser utilizados para representar sons. O formato SBI- Sound Blaster Instrument descreve como um instrumento é sintetizado a partir de um conjunto de parâmetros que controlam um sintetizador o qual utiliza a técnica de geração de sons por modulação em freqüência. O formato MOD - MODulation representa uma partitura de maneira semelhante a um arquivo MIDI, além da informação dos instrumentos utilizados na música, em uma representação digitalizada através da técnica de amostragem. Como dito anteriormente, é possível o armazenamento do som de uma maneira híbrida. Um exemplo é o caso do formato MOD, o qual contém duas partes :

- um banco de amostras digitalizadas;
- seqüência de informação descrevendo como e quando tocar as amostras.

O formato SNG - SoNG é equivalente ao formato MOD, porém representa os instrumentos através da parametrização para sintetizadores operando por modulação em freqüência. O formato SBK - Sound BanK permite o armazenamento de vários instrumentos representados no formato SBI num único arquivo.

Este trabalho foi desenvolvido utilizando um Modelo de Dados Orientado a Objetos específico, denominado MRO[Trai92], descrito na Seção 2.4, o qual incorpora conceitos de orientação a objetos, além de estender o conceito de tipos de dados com conceito de "característica de atributos". Essa nova característica agrupa os tipos de dados dos atributos que podem ser associados aos objetos, de acordo com o conjunto básico de operações que são permitidas sobre os atributos de cada característica. Por exemplo: os tipos de dados inteiro, real, real-de-precisão-dupla e outros podem ser manipulados pelas quatro operações aritméticas, e comparados por igualdade, ordenação, e assim por diante, constituindo a característica de números. Os tipos de dados cadeia-de-caracteres, texto_ASCII, texto_RTF- padrão Rich Text Format,entre outros podem ser concatenados, ter partes substituídas, e comparados por sub-

cadeias, constituindo a característica de texto. Essas operações são efetuadas mesmo entre dados representados de forma diferente (por exemplo, números com tipo de dado inteiro podem ser multiplicados por números com tipo de dado real, textos com tipo de dado RTF podem ser pesquisados por sub-cadeias com tipo de dados ASCII e outros, mas não é permitido somar um número com um texto).

Este trabalho explora esses conceitos quando aplicados à representação de partituras musicais. Nesse contexto, partitura passa a ser uma característica de dado, que pode ser representada de diferentes maneiras, cada uma sendo um tipo de dado. Por exemplo, um atributo que tem a característica partitura pode ser do tipo MIDI, ou ENC - ENCore, ou seja, independente do tipo usado o atributo não vai deixar de ser do tipo partitura. A mudança existe no armazenamento e manipulação do dado. Como visto anteriormente, há várias maneiras de se fazer isso, porém independente do meio escolhido, o atributo não deixa de ser do tipo partitura. Este trabalho explora a representação de partituras através do padrão SMF de arquivos MIDI [Gome93], [Hube95], [Stol93], [Ratt95] embora, dentro dos recursos propiciados pelo modelo de dados adotado, outras formas de representação de partituras possam ser igualmente suportadas. Para que uma nova característica de atributos possa ser suportada por um gerenciador de dados, além de definir pelo menos um tipo de dado, é necessário definir também o conjunto de operações básicas associadas a essa característica.

Este trabalho também define as operações de armazenagem, manipulação, busca e recuperação de dados necessárias, para que a característica partitura possa ser suportada por um gerenciador de dados orientado a objetos, no caso, o GEO.

4.2 Suporte para MIDI no Meta-Esquema

Um modelo de dados que tem a capacidade de modelar a si próprio é denominado **Meta-Modelo de Dados**. O MRO é um meta-modelo, e portanto pode representar suas próprias construções [Trai92]. O trabalho descrito estendeu MRO para o suporte à nova característica de partitura, e essa extensão pode ser representada utilizando os conceitos próprios de MRO, conforme visto na Figura 4.2. Esta figura descreve os tipos suportados pelo MRO, como visualização, regra, tempo, estrutura, gráfico, comentário e propriedade, descritos no Capítulo 2; juntamente com os novos tipos, BLOB, Partitura e Trilha, criados para o suporte a partituras. Na Figura 4.2 há o tipo **BLOB** que contém a estrutura necessária para o armazenamento de partituras e é representado como um **Atributo**; o tipo partitura indicado na Figura 4.2 como **Partitura** permite a representação de uma partitura, e o tipo trilha ilustrado como **Trilha**, define uma ou mais trilhas pertencentes a uma partitura.

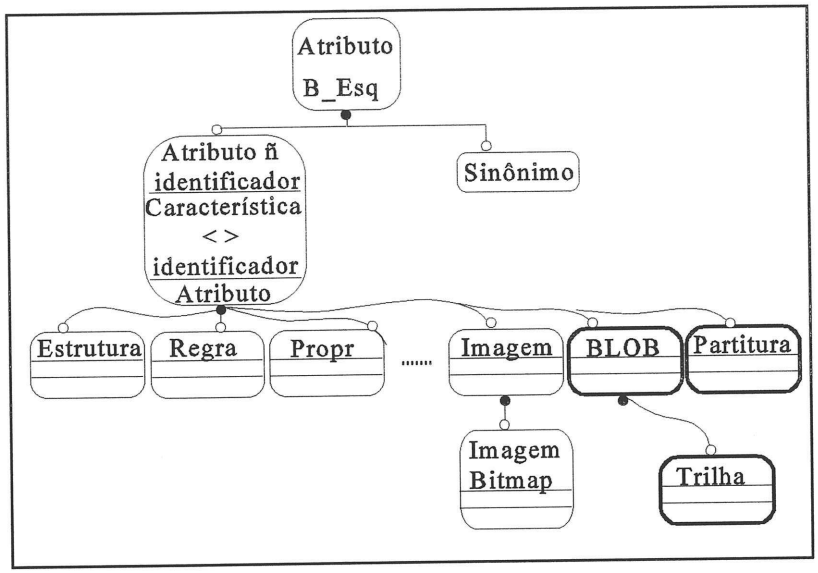

Figura 4.2 - Meta-Modelo de Dados

A Figura 4.3 mostra a modelagem da característica partitura utilizando a notação gráfica de MRO vista na Seção 2.4 através dos diagramas de representação. Nessa figura ilustra-se o meta-tipo objeto, representando um objeto que define o tipo de todos os objetos, e o meta-tipo atributo, representando um objeto que define o tipo de todos os atributos. O arco rotulado com Atribui/Atribuído_a representa o fato que atributos podem ser atribuídos a objetos. A

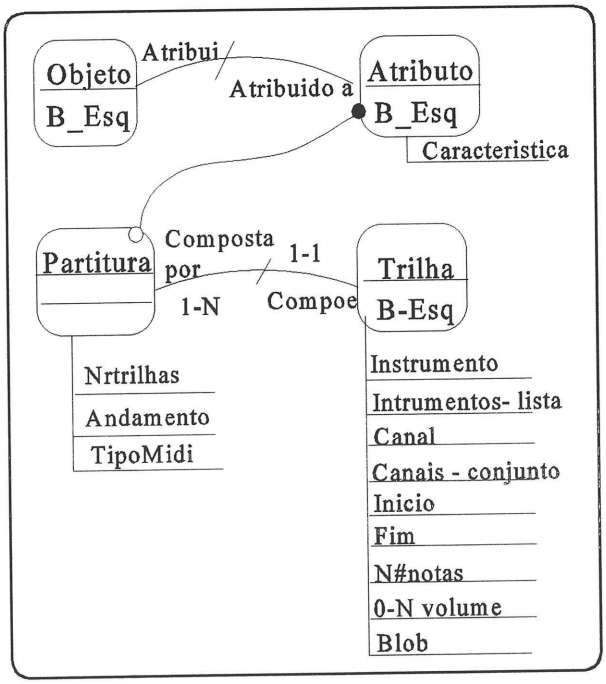

Figura 4.3- Representação de Partituras no MRO

notação  representa o fato de que Partitura é uma especialização de Atributos. Os atributos associados a objetos são representados por uma reta ligando o nome do tipo do atributo ao objeto, tal como a indicação que Atributos possuem uma Característica.

Tal como representado na Figura 4.3, uma partitura é composta por uma ou mais trilhas. As partituras têm os atributos: **Nrtrilhas** que informa o número de trilhas, **Andamento** relacionado ao andamento da música, e **TipoMidi** que possui o tipo do arquivo padrão MIDI, pois esse dado modifica a maneira como são tratadas as trilhas que compõem a partitura. Cada trilha possui os atributos: **Instrumento**, usado quando toda a trilha corresponde a dados de apenas um instrumento; **Instrumentos-lista** que é a lista de instrumentos usados quando a trilha concatena dados de diversos instrumentos; **Canal** correspondente à trilha; **Canais-conjunto** que é o conjunto de canais, quando a trilha utiliza mais de um canal, juntamente com a informação de instrumentos; **Início** e **Fim**, para indicar quando ocorre, respectivamente, o primeiro e o último evento da tripla; **N# notas** que possui as várias notas pertencentes a uma determinada trilha, para contar a polifonia máxima necessária para a trilha; **Volume** que indica o volume relativo das notas que soam nessa trilha em relação à partitura em geral; e **BLOB**, que corresponde à seqüência de eventos MIDI que constitui a trilha propriamente dita. Os atributos instrumento e canal são utilizados para arquivos tipo 1 e 2, para indicar a associação da trilha a um canal MIDI e a um instrumento. Já os atributos lista de instrumentos e conjunto de canais são utilizados para arquivos tipo 0. Um exemplo prático ocorre no caso da associação de um instrumento a um único canal, onde o canal 3 estará associado ao violão, ou quando a representação exigir mais de um instrumento relacionado a um único canal, onde há o canal 1 para os intrumentos piano e flauta como na Figura 4.4.

Dentre as informações descritas existem os eventos MIDI e os meta-eventos. Como descrito na Seção 3.5.8, os eventos MIDI correspondem aos relacionados diretamente com a execução musical e são encontrados no arquivo MIDI. No início dos eventos há o comando `note on` MIDI, indicando que será iniciada/tocada uma nota, em um determinado canal com uma certa intensidade. Então temos a nota a ser tocada e após esse comando aparece um `note off`, que interrompe a execução de uma nota em um determinado canal com uma certa velocidade de harmonia. Entre cada nota há o delta time que é o tempo de espera para o início da nota seguinte.

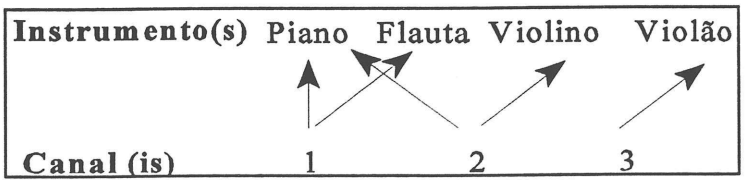

Figura 4.4 - Lista de Instrumentos/Canais

Além das informações descritas, a meta-modelagem da característica partitura inclui a definição de todo o padrão MIDI e todo o padrão SMF. Além disso, foi pré-definido também o padrão General MIDI. Assim, sempre que um usuário declara a intenção da utilização de atributos com a característica partitura em sua aplicação, o sistema incorpora automaticamente ao seu esquema de dados: a definição do tipo de objeto instrumento, visto na Figura 4.5; a definição de 196 objetos de tipo instrumento, um para cada um dos instrumentos do padrão GM; a definição de 11 tipos de atributos correspondentes aos meta-eventos padronizados pelo padrão GM; e a definição de 12 tipos de atributos correspondentes aos controladores de parâmetros musicais definidos pelo Padrão MIDI [Stol93]. Todos os tipos de atributos definidos têm sua sintaxe automaticamente estendida para que possam ser associados a objetos de todos os tipos que o usuário definir que podem ser associados atributos com a característica partitura.

A definição dos objetos tipo instrumento e tipos de atributos permite que o sistema possa extrair informações dos arquivos padrão MIDI, e armazená-las como atributos dos objetos. Isso facilita as operações de busca envolvendo o conteúdo das partituras, tais como consultas do tipo: "Quais as músicas que utilizam determinado instrumento?".

As consultas envolvendo a localização de seqüências de notas são implementadas de maneira semelhante à busca de sub-cadeias em textos. Assim, as mesmas operações de comparação de cadeias (seqüências de caracteres) são permitidas envolvendo seqüências de notas.

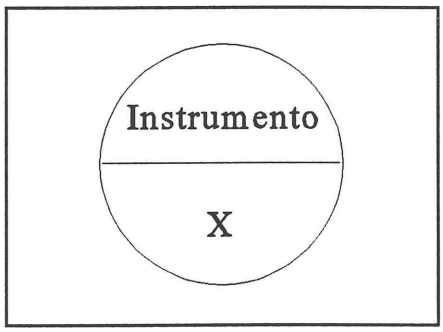

Figura 4.5 -Definição de Tipo de Objeto Instrumento

As operações de recuperação de partituras são feitas de duas maneiras: geração de arquivos MIDI, e geração de um "response set" em memória. A geração de arquivos é feita individualmente, arquivo por arquivo. A geração de um "response set" permite que um conjunto de atributos sejam gerados, um a um, e submetidos individualmente a tratamento pelo aplicativo. Em ambos os casos, a recuperação dos dados segue o conjunto de atributos que o usuário indica que devem ser incluídos. É interessante notar que com esse enfoque, é possível atribuir dados a partituras manualmente, ou seja, o usuário manipula os atributos textuais de uma partitura independentemente da partitura. Quando uma partitura é recuperada, os atributos indicados somente são incluídos na estrutura recuperada caso eles tenham um valor definido.

4.3 Suporte das Definições Padrão no Meta-Esquema

Um arquivo MIDI usualmente armazena informações a respeito de uma única música. Os eventos MIDI e os meta-eventos contém todas as informações necessárias para compor a partitura dessa música, sendo que os eventos exclusivos do sistema adequam a execução da música para um instrumento (ou um conjunto de instrumentos) específico, e portanto estes não serão considerados neste trabalho como integrantes de uma partitura.

Dispondo-se das partituras em arquivos, sua representação em um banco de dados pode ser feita de duas maneiras: armazenando-se o arquivo de partituras para "dentro" do banco de dados, ou mantendo-se as partituras nos arquivos originais e gravando-se o nome do arquivo no banco de dados, mantendo assim apenas uma referência para a partitura. O último enfoque não garante consistência das informações armazenadas no banco de dados, pois o usuário pode modificar o arquivo sem que o gerenciador do banco de dados tenha conhecimento disso. O primeiro enfoque é então o adotado neste trabalho.

Uma outra decisão de projeto corresponde ao tratamento que deve ser dado na armazenagem da partitura, a qual pode ser vista como um objeto do banco de dados, ou como o valor de um atributo. Este trabalho adota a segunda opção, pois permite associar partituras a qualquer objeto. Por exemplo, caso o usuário queira armazenar músicas como objeto, basta criar explicitamente o tipo de objeto música, o qual pode ter um atributo chamado partitura, monovalorado com característica de partitura, e cujo valor é a partitura propriamente dita. Dessa maneira, a característica partitura é suportada restringindo-se às informações puramente musicais, ou seja, o valor de um atributo de característica partitura inclui apenas as informações correspondentes aos eventos MIDI de um arquivo SMF. Os meta-eventos passam a ser outros atributos, com características diferentes de partituras, associadas aos objetos nos quais a partitura está atribuída, como visto no Quadro 3.4.

Assim como a partitura (eventos MIDI), os meta-eventos são considerados atributos do tipo partitura. Porém, como dito anteriormente, existem as definições padrão (meta-eventos) do arquivo MIDI que poderiam ser complementadas com outras informações. Como foi descrito no Capítulo 3, existem vários meta-eventos próprios do arquivo MIDI, e além desses foram criadas informações extras que ampliam o limite de consulta para o usuário.

Por exemplo, são guardadas informações, tais como, o nome do autor, estilo musical, entre outros, permitindo desde consultas como: "Quais as músicas que utilizam piano?", que usa o meta-evento `nome da música` existente no arquivo MIDI, até: "Quais são as músicas de Tom Jobim?", que usa o meta-evento extra `nome do autor`, ou seja, não existente no arquivo MIDI.

Outro exemplo, permite que objetos de tipo música tenham além do atributo `partitura`, um atributo `data_de_publicação` de característica data, e atributos `autores` e `letra` de característica texto.

4.4 Operações de E/S de Partitura na LAMRO

O GEO permite que sejam definidos esquemas, onde são informados os vários objetos com seus relacionamentos e atributos, montados e baseados no MRO. Além disso, ele permite a inclusão de dados no banco de dados [Figu96].

Com o objetivo de definir e manipular os dados foi desenvolvida a Linguagem de Acesso ao MRO - LAMRO [Corr92].

A LAMRO visa estabelecer uma comunicação de "alto nível" entre o usuário e o MRO, tendo em vista permitir sua utilização de forma interativa.

Para definir esta linguagem foram consideradas algumas características que pudessem guiar a sua definição, entre elas:

- mapear os conceitos definidos no MRO em construções sintáticas e suas respectivas ações semânticas;
- auxiliar no aprendizado dos conceitos do Modelo;
- ser de fácil utilização;
- permitir sua implementação segundo o ambiente operacional do GEO.

Dentro da classificação adotada para os comandos, dividiu-se cada conjunto de comandos naqueles que manipulam o esquema e naqueles que manipulam o banco de dados propriamente dita. Os comandos da linguagem atuam sobre os elementos do modelo e suas instâncias tais como : objetos, relacionamentos, atributos, colônia, subtipo e supertipo. Os comandos foram classificados de acordo com a finalidade para os quais foram definidos:

- inclusão - inclui a definição de um novo tipo, ou a instância de um elemento do modelo;
- eliminação - elimina algum elemento incluído;
- substituição - altera alguns dos elementos;
- mostra - exibe informações sobre os diferentes elementos pertencentes ao modelo.

Os comandos de inclusão permitem introduzir elementos como objeto, colônia, relacionamento, atributo.

Com este trabalho foi incluída a opção de introduzir atributos do tipo partitura ao nível do esquema do banco de dados, permitindo que se manipule a partitura ao nível do banco de dados.

A LAMRO foi estendida no que se refere aos comandos de acesso ao banco de dados permitindo a importação, exportação, listagem, posicionamento, avanço e execução da partitura através dos comandos: `importa`, `exporta`, `lista`, `posiciona`, `avança` e `toca` respectivamente.

Assim como na LAMRO existente, os comandos acima podem estar vinculados a um objeto ou relacionamento corrente.

Com base no exemplo da Figura 4.6, são aplicados os comandos existentes na LAMRO para definição e manipulação de partituras, permitindo o armazenamento e consulta do atributo partitura. Estes comandos são mostrados nas 4.7 a 4.14.

Considerando que exista um objeto País que é instanciado com X, e possua os atributos: Presidente, que informa o nome do presidente aceitando dados do tipo caracter; Hino Nacional, que armazena o hino nacional, sendo portanto um atributo do tipo partitura; Músicas, que contém as principais músicas referentes àquele país, sendo definidas como atributo do tipo partitura, já que armazena dados MIDI; e Autor do hino nacional, que informa o nome do autor do hino nacional, aceitando dados do tipo caracter.

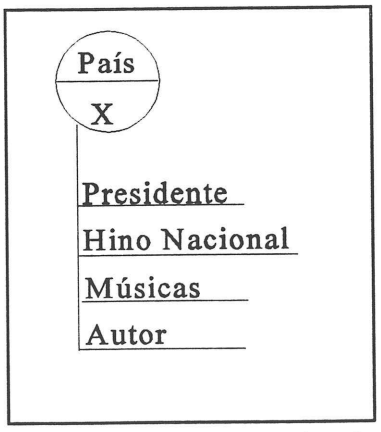

Figura 4.6 - Exemplo de Esquema País

Importa

Permite a inclusão de um atributo de tipo partitura no banco de dados.

A inclusão de uma partitura no banco de dados é feita através da importação desta, utilizando o comando:

Sintaxe

IMPORTA <Tipo de Partitura> <Nome do Arquivo>;

Exemplo

Incluir o Hino Nacional do Brasil:

IMPORTA <hino nacional> brasil.mid;

Figura 4.7- Importação de uma Partitura

Obs.: O nome do arquivo MIDI tem extensão mid; se a extensão não for informada, esta será colocada como valor *default*.

Exporta

Realiza a exportação de um atributo de tipo partitura para um arquivo MIDI. A transferência de uma partitura para um arquivo MIDI pode ser feita através do comando:

Sintaxe

EXPORTA <Tipo de Partitura> **PARA** <Nome do Arquivo>;

Exemplo

Exportar as músicas para o arquivo franca.mid:

EXPORTA <musicas> **PARA** <franca.mid>;

Figura 4.8- Exportação de uma Partitura

Liga

Cria uma lista de saída através da ligação de atributos do tipo partitura. Dentre os atributos existem os meta-eventos do arquivo MIDI e os meta-eventos incluídos. Este comando é muito útil para listar vários atributos distintos.

Sintaxe

LIGA PARTITURA [Meta-Evento + Extra]

Exemplo

Listar o autor do hino:

```
LIGA PARTITURA [Meta-Evento + Extra];
LISTA;
```

Figura 4.9- Ligação do Atributo Partitura

Desliga

Desvincula a lista criada pelo comando LIGA, ou seja, os atributos perdem a ligação definida. A ligação desvinculada será relacionada ao tipo de partitura corrente.

Sintaxe

DESLIGA PARTITURA

Exemplo

```
DESLIGA PARTITURA
```

Figura 4.10- Desativação de uma Lista de Partituras

```
Lista
     Exibe todos os nomes dos atributos do tipo partitura.
Sintaxe
     LISTA
Exemplo
     Exibir os nomes do autores do hino.
     LIGA PARTITURA autor do hino
          LISTA
```

Figura 4.11- Listagem dos Atributos de uma Partitura

```
Posiciona
     Posiciona no atributo de partitura especificado.
Sintaxe
     POSICIONA <Tipo de Partitura>
Exemplo
     Posicionar nas músicas.
     POSICIONA musica
```

Figura 4.12- Posicionamento de uma Partitura

```
Avança
     Este comando permite o avanço para o tipo de partitura seguinte.
Sintaxe
     AVANCA
Exemplo
     POSICIONA hino nacional
          AVANCA
```

Figura 4.13-Avanço de um Atributo da Partitura

```
Toca
     Permite a execução do atributo partitura corrente.
Sintaxe
     TOCA
Exemplo
     Tocar o hino nacional
     POSICIONA hino nacional
          TOCA
```

Figura 4.14- Execução de uma Partitura

Um exemplo mais detalhado utilizando alguns dos comandos descritos acima, pode ser visto no apêndice desta dissertação.

4.5 Considerações Finais

Neste capítulo foram mostrados os conceitos utilizados para que o tipo de dado partitura fosse representado como uma característica no MRO. Essa expansão foi feita criando-se dois conceitos novos: **BLOB**, que permite a representação da informação sem que se saiba o seu contéudo e a **partitura**, que interpreta um BLOB como sendo do tipo MIDI. Essa interpretação pode ser feita pelo reconhecimento do formato interno de tratamento do MIDI ou pelo reconhecimento do tipo de informação colocada na estrutura de dados através de um meta-esquema que suporta qualquer aplicação além do MIDI.

O suporte para representar áudio incluiu:

- padrão MIDI, que permite a armazenagem de informação musical;
- padrão GM, que permite a representação de instrumentos de maneira padronizada;
- padrão SMF, que é a representação do arquivo MIDI, com a informação de temporização.

Além disso, foi proposta uma linguagem básica para manipulação da informação de partitura, a LAMRO. Para atender a esse propósito, foi necessária a extensão da LAMRO, nos comandos de acesso ao banco de dados.

CAPÍTULO 5
Implementação do Suporte de Partituras

Neste capítulo é descrita a implementação da característica "partitura musical", suportada em nível de modelagem pelo MRO (descrito no capítulo anterior), e agora no GEO permitindo que esta seja manipulada pelo usuário.

5.1 Arquitetura do GEO

O GEO foi implementado baseando-se no MRO. Sua implementação se deu com o objetivo de validação prática e estudo de como os conceitos de orientação a objetos podem ser implementados em um sistema de *software* real. O principal objetivo de seu desenvolvimento é permitir a construção de banco de dados, que suportem a representação de informações dos sistemas de apoio a projetos de engenharia, CAD, de ambientes de suporte ao desenvolvimento de sistemas, CASE, e de sistemas de coleta e tratamento de dados científicos/estatísticos.

Para implementar o gerenciador de forma que este possa manipular eficientemente a base extencional e a base intencional, o GEO foi dividido em cinco módulos, como pode ser visto na Figura 5.1:

1. **Núcleo de Acesso a Registros**: consiste de um sistema de gerenciamento de arquivos lógicos, fornecendo suporte aos demais módulos. É composto pelo Subsistema de Gerenciamento de Registros - SGR, Subsistema de Gerenciamenteo de Arquivos - SGA e Subsistema de Gerenciamento de Memória - SGM. Estes módulos realizam todo o gerenciamento de memória secundária, simulação de uma memória *cache* para disco em memória, e controle de acesso e proteção dos dados, através do gerenciador de transações.

2. **Subsistema Básico de Gerenciamento**: suporta a manipulação de objetos sem restrições de ordem semântica. Envolve o tratamento de identificadores de objetos, manipulação de composições de objetos, contexto e navegação no banco.

3. **Subsistema Básico de Gerenciamento de Relacionamentos**: suporta a manipulação de relacionamentos (binários ou triplos) definidos pelo usuário. O GEO gerencia relacionamentos intrínsecos e relacionamentos extrínsecos: os intrínsecos são aqueles cuja semântica é conhecida pelo gerenciador, tal como as hierarquias de generalização e de composição, sendo gerenciadas pelo Subsistema Básico

de Gerenciamento de Objetos; os extrínsecos são aqueles cuja semântica é definida pelo usuário e são gerenciados por este módulo em questão.

4. **Subsistema Básico de Gerenciamento de Atributos**: permite a manipulação de atributos com características de propriedades e comentários, tanto para objetos quanto para relacionamentos.

5. **Gerenciador de Esquemas de Dados - GED**: consiste nas primitivas de manipulação, acesso e verificação de esquemas de dados, as quais são ativadas pelos demais módulos.

Correspondendo a cada **subsistema de gerenciamento básico** está sendo realizada a implementação de um **subsistema de gerenciamento semântico**. O conjunto de subsistemas de gerenciamento básico é denominado **GEO/MRO-Básico** e o conjunto de subsistemas de gerenciamento semântico é denominado **GEO/MRO-Semântico**, como visto na Figura 5.1.

O GEO/MRO-Básico efetua todas as operações de acesso efetivo no banco de dados, tal como criar ou eliminar objetos, relacionamentos, entre outros. Já o GEO/MRO-Semântico tem como objetivo realizar as mesmas operações, porém avaliando não apenas as regras sintáticas estáticas como é feito agora pelo GEO/MRO-Básico, mas também as regras semânticas, que dependem do ambiente instântaneo dos dados no banco de dados [Camo92].

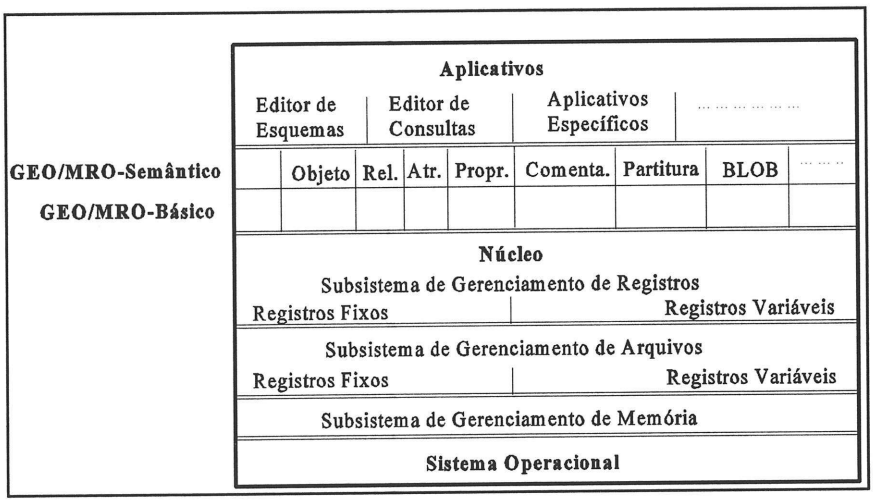

Figura 5.1-Arquitetura do GEO

5.2 Arquitetura do Suporte de Partituras no GEO

Para suportar partituras em nível físico foi necessário estender os conceitos definidos no MRO, apresentados na Seção 2.4, conforme vistos no Capítulo 4, permitindo que O MRO armazene os atributos do tipo partitura. Esse suporte é estendido igualmente no GEO para a manipulação da partitura pelo usuário, resultando na *interface* de acesso, através de um **editor de partituras** , cuja arquitetura

básica é mostrada na Figura 5.2. Como descrito anteriormente na Seção 5.1, o GEO foi dividido em cinco módulos, entre os quais foram alterados:

- Subsistema Básico de Gerenciamento de Atributos - permitindo a manipulação de atributos do tipo partitura.

- Gerenciador de Esquemas de Dados - alterando as primitivas de acesso ao esquema para reconhecimento do tipo de atributo partitura.

O **editor de partituras**, visto na Figura 5.2, é responsável pela manipulação do atributo de tipo partitura, e envolve os seguintes processos:

- Entrada: a entrada de dados inclui arquivos MIDI completo - **SMF**, mostrado na Figura 5.2, ou seja, contém as partituras, os meta-eventos e os eventos exclusivos do sistema.

- Processamento: feito pelo **MRO** e **GEO** através do tratamento de atributos do tipo partitura e a codificação de rotinas de alto nível respectivamente, para a manipulação pelo usuário na **BD - Base de Dados** ilustrada na Figura 5.2. Dentre estas rotinas, temos por exemplo a **bdpar**, vista na Figura 5.2, que faz a leitura do tipo de partitura corrente. Para isso, o GEO recebe arquivos SMF, faz o tratamento das informações, filtrando as que realmente serão usadas, no caso os eventos MIDI e os meta-eventos, e retorna os arquivos "limpos" a serem manipulados, como é demonstrado no fluxo do **MRO/GEO** para o ***driver***. Essa operação de filtragem será melhor explicada no Seção 5.3.

- Saída: permite a execução da música, através do acionamento de ***drivers***, que são "pontes"entre o *hardware* que cria o som da nota e o programa que toca a música MIDI. Para isso, os *drivers* recebem os arquivos SMF "limpos" do GEO, e através de rotinas executa as músicas solicitadas pelo usuário. A execução da música é feita com o auxílio de um dispositivo de saída, por exemplo, um amplificador.

Esse processo, por sua vez, interage com o **usuário**, visto na Figura 5.2, através de solicitações enviadas para o sistema e respostas recebidas deste, permitindo que sejam feitas consultas do tipo: "Selecione todas as múscias de Tom Jobim".

Como visto, esse editor atende a dois propósitos: por um lado permite que o usuário solicite efetivamente as operações de armazenagem, manipulação, busca e recuperação de partituras; por outro lado foi o protótipo que concretizou os pressupostos e a estrutura de dados proposta no Capítulo 4, além de ser o mecanismo através do qual as idéias e a implementação realizadas foram validadas. É importante notar que os conceitos descritos neste trabalho não estão restritos ao escopo deste Editor de Partituras, porém o mesmo pode ser utilizado como um guia para o desenvolvimento de outros aplicativos que utilizem a armazenagem de partituras em bancos de dados orientados a objetos.

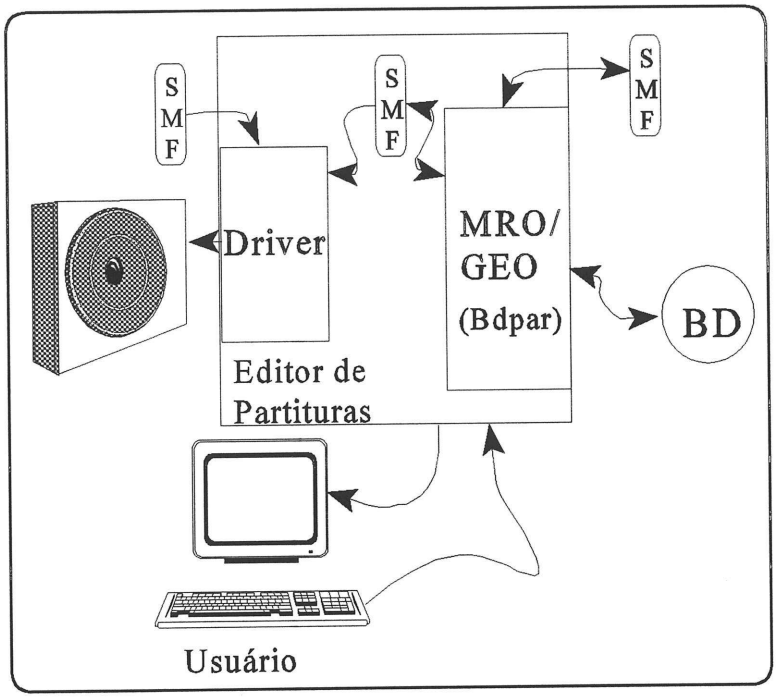

Figura 5.2- Arquitetura do *Software*

5.3 Inicialização do Esquema de uma Aplicação Habilitada para MIDI

O reconhecimento pelo GEO de atributos do tipo partitura inclui um processo que envolve desde a **inicialização do esquema** até a **manipulação das partituras**. O processo ainda considera a **definição do esquema** e a **definição das operações de gerenciamento** feitas no OPRGER, que serão explicadas em detalhes em seguida.

O GEO [Trai92] permite que arquivos padrão MIDI sejam armazenados diretamente no banco de dados como atributos de característica partitura, bem como permite recuperar tais atributos recriando arquivos padrão.

Na Figura 5.3 há um detalhamento em nível físico do processo da Figura 5.2 incluindo desde o fluxo de entrada de uma partitura em nível de reconhecimento no esquema, até a manipulação desta através do gerenciador de objetos.

O fluxo engloba a **definição do meta-esquema** feita para o suporte da definição do tipo de atributo partitura, como visto na Figura 5.3, para posteriormente ser reconhecida e criada um banco de dados ou arquivo no **OPRGER**. Este conseqüentemente inclui: a **criação do meta-esquema básico**, responsável pelo reconhecimento dos tipos suportados pelo MRO e a **criação do meta-esquema de partitura**, que foi incluído para permitir o reconhecimento do tipo partitura, descritas detalhadamente em seguida.

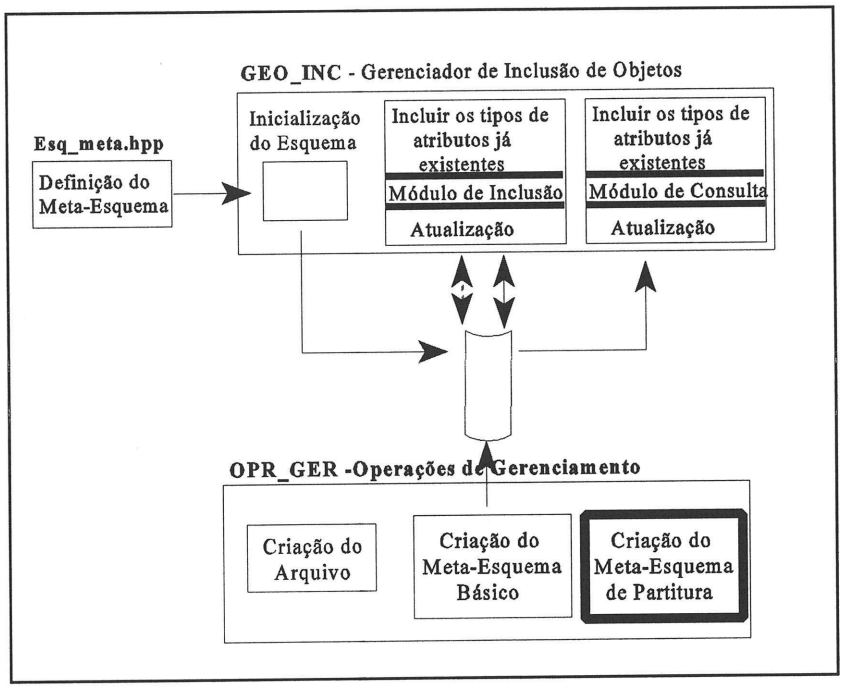

Figura 5.3 - Meta- Esquema para Partitura

Após a criação do banco de dados, é necessária a manipulação dos dados através do **GEO**, como visto na Figura 5.3. No GEO é feita inicialmente a **inicialização do esquema**, permitindo o reconhecimento de definições do tipo partitura. Para a manipulação foram construídos os **módulos de atualização** e os **módulos de consulta** que permitem a inclusão e a manipulação de atributos do tipo partitura respectivamente, que serão descritos em seguida mais detalhadamente.

Para que seja feito o armazenamento e a recuperação de atributos, é necessário que este seja definido no meta-esquema do MRO, ou seja, é necessário preparar o banco de dados para que o atributo partitura seja manipulado. A preparação do banco de dados inclui o reconhecimento de atributos do tipo partitura através de informações referente a este, como por exemplo, a definição do tipo de instrumentos.

Na **definição do meta-esquema** é feita a preparação para a definição do tipo de atributo partitura. Isso foi incluído no código fonte `esq_meta.hpp`, onde estão todas as definições dos objetos aceitos pelo MRO.

As operações de gerenciamento são controladas pelo aplicativo **OPRGER**, onde é feita a criação do arquivo do banco de dados. Para que este seja reconhecido, é necessário que os tipos sejam válidos. Por isso é que temos a criação do **meta-esquema básico** e o meta-esquema de partitura. Na criação do meta-esquema básico existem as definições dos tipos suportados pelo MRO como sinônimo, regra, estrutura de dados, som, visualização, gráfico, tempo, comentário, imagem e na criação do **meta-esquema de partitura** foram feitas as inclusões necessárias para permitir o reconhecimento do tipo partitura.

O OPRGER é o operador de gerenciamento responsável, como o próprio nome diz, pelo gerenciamento do banco de dados criada no MRO. Através deste aplicativo é possível :

- inserir registros;
- modificar registros;
- listar registros;
- criar um banco de dados MRO;
- finalizar modificações.

A criação do banco de dados requer inicialmente a inserção de um registro, e depois a sua criação. Esse processo engloba um conjunto de comandos que podem ser vistos em [Figu96].

Após a definição do banco de dados, é criado um arquivo com a estrutura vazia, na qual estão:

- colônia global (Besquema) que possui as informações passadas via OPRGER;
- colônia esquema que possui a estrutura do meta-esquema, incluindo as definições para atributo do tipo partitura, ou seja, a **criação do meta-esquema básico** e a **criação do meta-esquema de partitura**.

Após isso, podemos considerar que o banco de dados foi criada com a definição dos tipos básicos e as definições MIDI, permitindo o reconhecimento do tipo de atributo partitura.

Agora é necessária a inicialização do banco de dados, ou seja, preparar as estruturas e as colônias do sistema para operação com MIDI. Uma vez inicializada o banco de dados podemos guardar as informações no banco de dados.

Os arquivos padrão MIDI [Stol93] possuem muitas informações que não correspondem às informações musicais. Apenas os eventos MIDI são necessários para a efetiva representação de uma partitura. Assim, quando um arquivo é recebido pelo gerenciador de objetos, o mesmo é separado em suas trilhas componentes, e cada uma destas é filtrada, para que somente permaneçam os eventos MIDI,

como é mostrado na Figura 5.4. Os eventos exclusivos de sistema são descartados, pois não fazem parte da música em si, mas de uma configuração de instrumentos. Os meta-eventos são retirados, e submetidos ao próprio gerenciador de objetos como atributos `texto` com tipos pré-definidos para serem associados ao mesmo objeto onde a partitura está sendo associada. Assim, os atributos partitura são armazenados com um conjunto de informações padronizadas, que correspondem estritamente ao conteúdo musical de um arquivo padrão MIDI. Os outros atributos são armazenados como atributos normais, e após a finalização de uma operação de armazenagem, não são distintos dos que um usuário poderia armazenar manualmente.

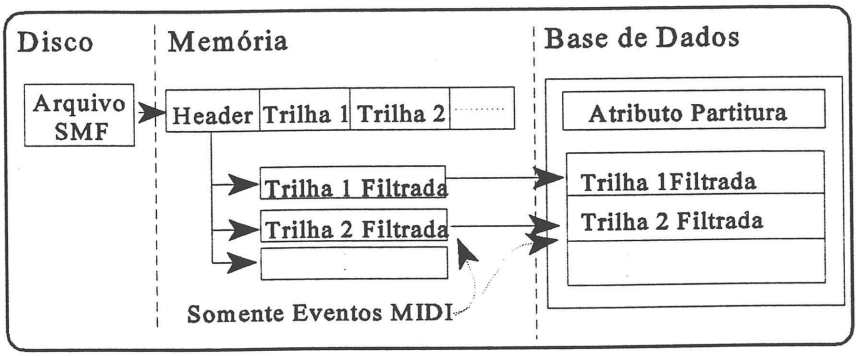

Figura 5.4- Estrutura de Armazenagem de Partitura

Como o padrão MIDI define um conjunto de informações padronizadas, para que uma aplicação possa ser habilitada a manipular arquivos MIDI, ela deve estar preparada para isso, e isso reflete-se em seu esquema de dados: toda aplicação habilitada para tratar o padrão MIDI deve ter o esquema de dados do padrão MIDI como parte de seu esquema de dados, que é mostrado na Seção 4.3 como um meta-modelo. No entanto, a aplicação que estiver habilitada pada suportar o padrão MIDI deve também ser capaz de reconhecer um conjunto de atributos já pré-definidos, que podem ser associados a qualquer objeto da aplicação que possa ter alguma partitura associada. A maior parte desses atributos estão relacionados aos meta-eventos, uma vez que, embora sejam armazenados num arquivo MIDI, não correspondem de fato à informação básica de uma partitura. Quando os meta-eventos são "removidos" das trilhas de uma partitura, as informações que eles representam são passadas para atributos com características diferentes de partitura, associados ao mesmo objeto onde o atributo com característica de partitura está associado. O tipo desses atributos é um tipo pré-definido para esquemas de dados de aplicações habilitadas para suportar o padrão MIDI, mostrado na Figura 4.3.

5.4 Operações de Importação e Exportação de Partituras

Como visto no Capítulo 4, a LAMRO foi estendida para suportar as operações de importação e exportação de partituras, permitindo assim a inclusão e a consulta de partituras respectivamente.

Isso é feito com a codificação de vários módulos, mostrados no Quadro 5.1, e utilizado através do aplicativo GEO_INC - Gerenciador de Inclusão de Objetos. Através deste gerenciador foram analisados dois tipos de operações:

- inclusão - inclusão de dados do tipo partitura;
- consulta - consulta através de parâmetros dos dados do tipo partitura.

As rotinas responsáveis tanto pela inclusão como pela consulta são mostradas no Quadro 5.1, e uma descrição da funcionalidade de cada uma pode ser vista no Quadro 5.2.

Quadro 5.1 - Descrição das Rotinas de Inclusão e Consulta

Tipo de Atributo	Inclusão	Leitura	Posiciona	Exclusão	Exporta	Avança
Partitura	CTPAR CTTRL	BDPAR BEPAR BNPAR BFPAR	PNPAR	RNPAR	FTPAR FFPAR	VNPAR
Comentário	CTCOM	BDCOM	PNCOM TACOM	RECOM RNCOM	-	VECOM VNCOM
BLOB	CTBLOB	BDBLOB BEBLOB	TABLOB	-	-	VNBLOB
Trilha	I_CTPAR	-	PNTRL	RNTRL	-	VNTRL

Quadro 5.2 - Função das Rotinas de Inclusão e Consulta

Inclusão	
CTPAR (char *FnMIDI)	Permite a inclusão de uma partitura, informando o nome do arquivo MIDI a ser incluído.
CTTRL (No.Trilha, *trilha)	Escreve a trilha da partitura corrente
CTCOM (Coment, CTA)	Coloca uma linha de comentário de tipo "CTA" no objeto/relacionamento corrente. A seleção do tipo de dado do comentário é determinada pelo tipo do comentário.
I_CTPAR (trilha)	Lê um arquivo de ponteiros e executa todas as notas naquele instante, armazenand-as na pilha de notas.
CTBLOB (*Buffer, tamanho,CTA)	Coloca um Registro BLOB do tipo CTA com o conteúdo indicado no OBJ/REL corrente.
Leitura	
BDPAR (CTA, No.Trilhas, Tipo de arquivo)	Obtém o tipo da partitura corrente.
BEPAR (No.Trilha, *trilha)	Obtém número da trilha.
BFPAR (CTA, No.Trilhas, Tipo de arquivo, No.Trilha, *trilha)	Obtém dados da partitura corrente.
BNPAR (CTA)	Obtém código da partitura corrente.
BDCOM (coment, CTA)	Lê a linha do comentário corrente, informando qual é o código do tipo do comentário, e qual seu valor.
BDBLOB (*CTA, *tamanho)	Lê o BLOB corrente, informando qual o código de tipo atributo (CTA) e qual o tamanho da área de dados.
BEBLOB (tamanho, char *Buffer)	Lê o BLOB corrente, e passa o conteúdo do BLOB para o Buffer cujo endereço foi indicado.

Quadro 5.2 - Função das Rotinas de Inclusão e Consulta (Continuação)

Posicionamento	
PNPAR (CTA)	Posiciona na primeira partitura deste tipo.
PNCOM (Coment, CTA)	Posiciona na primeira linha do comentário do tipo CTA que possua uma sub-cadeia com valor indicado, do objeto/relacionamento corrente.
TACOM (CTA)	Posiciona na primeira linha de comentário do tipo CTA do objeto/relacionamento corrente. Caso CTA seja zero, então posiciona na primeira linha de comentários em geral do objeto/relacionamento corrente, sem importar qual o seu tipo.
TABLOB (CTA)	Esta rotina posiciona no primeiro BLOB de tipo CTA no OBJ/REL corrente.
PNTRL (No.da Trilha)	Posiciona na trilha indicada.
Exclusão	
RNPAR ( )	Retira partitura corrente.
RECOM ( )	Retira todas as linhas do comentário corrente.
RNCOM ( )	Retira a linha corrente do comentário corrente. Desativa comentário corrente.
RNTRL ( )	Retira linha da partitura corrente.
Exportação	
FFPAR (Tipo do arquivo, atributo[ ])	Inclui os meta-eventos indicados.
FTPAR (Tipo do arquivo)	Exporta a partitura corrente para o arquivo indicado.

Quadro 5.2 - Função das Rotinas de Inclusão e Consulta (Continuação)

Avança	
VNPAR ( )	Avança partitura.
VECOM ( )	Avança para o próximo comentário na seqüência de comentários do objeto/relacionamento.
VNCOM ( )	Avança para a próxima linha do comentário corrente.
VNBLOB ( )	Avança para o próximo BLOB na seqüência de BLOBs do OBJ/REL corrente.
VNTRL ( )	Avança a trilha.

5.4.1 - Módulos de Inclusão

No módulo de inclusão de dados, é permitido incluir dados dos tipos suportados pelo MRO, entre eles: sinônimos, comentário, regra, procedimento, tempo, estrutura de dados, gráfico, imagens, som e visualização. Além desses tipos é possível a inclusão de dados do tipo partitura através da operação de importação.

A operação de importação permite atribuir valores a atributos partitura. Isso é feito através do método implementado pela rotina `ctpar`, que faz o tratamento de uma partitura pelo reconhecimento dos dados de um arquivo MIDI. E através da rotina interna i_ctpar, que faz o tratamento de uma trilha, verificando se os dados são eventos MIDI para que sejam armazenados e manipulados; eventos exclusivos do sistema que são ignorados e meta-eventos que se transformam em atributos no banco de dados.

5.4.2- Módulo de Consulta

A consulta de uma partitura faz parte da linguagem de manipulação dos dados, permitindo ao usuário realizar operações de consulta e operações de transformação de partituras, como:

- exportação - constrói na cadeia o valor de atributo codificado, através da rotina **ftpar**, vista no Quadro 5.2.

- alteração - permite que seja feita uma modificação na partitura, por exemplo, alterar a seqüência de notas.
- posicionamento:

  bnpar - obtém o código da partitura corrente.

  bdpar - obtém o tipo da partitura corrente.

  bfpar - obtém os dados da partitura corrente.

- avanço - avança para a partitura seguinte através da rotina **vnpar**, mostrada no Quadro 5.2.
- exclusão - retira a partitura corrente pela rotina **rnpar**, descrita no Quadro 5.2.
- mostra - listagem do arquivo MIDI.

O mesmo foi feito para a estrutura definida para o armazenamento da trilha de uma partitura, ou seja,o BLOB. As rotinas de inclusão e consulta englobam:

- importação - coloca um registro BLOB do tipo CTA- Código do Tipo de Atributo indicado no objeto/relacionamento corrente, através da rotina **ctblob**.
- posicionamento - posiciona no primeiro BLOB de tipo CTA no objeto/relacionamento corrente, através da rotina **tablob**.
- avanço - avança para o próximo BLOB na seqüência de BLOBs do objeto/relacionamento corrente, através da rotina **vnblob**.
- leitura :

  beblob- lê o BLOB corrente, e passa o conteúdo do BLOB para o buffer cujo endereço foi indicado.

  bdblob -lê o BLOB corrente, informando qual o CTA e qual o tamanho da área de dados.

O tratamento da trilha de uma partitura também possui rotinas de inclusão e consulta, entre elas:

- escrita - escreve a trilha da partitura corrente através da **cttrl.**
- posicionamento - posiciona na trilha indicada, através da rotina **pntrl**.
- avanço - avança a trilha , através da rotina **vntrl**.
- exclusão - exclui a trilha corrente, através da rotina **rntrl**.

5.5 Considerações Finais

Este capítulo descreveu a implementação no GEO dos requisitos necessários à manipulação de partituras. Esta implementação foi feita utilizando a linguagem C em ambiente Windows, embora possa ser independente do ambiente.

Assim como o tratamento no MRO, descrito no capítulo anterior, que prevê a inclusão de características novas, o GEO também apresenta esta flexibilidade de inclusão de módulos responsáveis pelo gerenciamento das características de partitura. Para isso foi desenvolvido um módulo novo com o objetivo de suportar partituras, que trabalham com tipo de dado MIDI, e outro para suportar BLOB, que é mais genérico, podendo suportar, por exemplo, característica imagem.

O trabalho realizado "enfocou" o armazenamento e não a execução da partitura. Embora existam alguns módulos responsáveis pela execução da partitura, verificando assim se o armazenamento está correto, existem aplicativos que são próprios para essa execução. O banco de dados simplesmente armazena, recupera e manipula as informações musicais, ou seja, todo acesso feito fica independente da execução da música. Por outro lado, o sistema fica altamente portável pois não depende de uma configuração particular de recurso de áudio.

CAPÍTULO 6
Conclusão

Neste capítulo são descritas algumas decisões de projeto realizadas durante o trabalho, os objetivos alcançados e sugestões de pesquisas futuras para o Editor de Partituras.

6.1 Decisões de Projeto

É importante registrar neste trabalho a justificativa para algumas das decisões que foram tomadas em diversas fases de desenvolvimento do projeto, as quais são apresentadas a seguir.

6.1.1 Por que o Formato MIDI ?

Os formatos descritos no Capítulo 3 levaram à escolha do formato MIDI, pois ele é o único padrão para representação de partituras. Os outros formatos estudados, ou não representam simplesmente a partitura mas também outras informações, ou são formatos proprietários de algum fabricante em particular, tais como os formatos SNG (Creative Labs) e WRK (Cakewalk).

Existem outros padrões para a armazenagem de áudio, como por exemplo, os arquivos WAV, descritos anteriormente, mas nesse caso há a representação do som digitalizado, ou seja, a gravação através de ondas sonoras e não de partituras. Neste trabalho, estamos interessados na recuperação de informações sobre a música. Isso não ocorre com os arquivos padrão WAV, já que a informação musical é gravada através da digitalização de ondas sonoras. Para que a recuperação de uma partitura seja feita a partir da gravação do áudio gerado por sua execução, é necessária a criação de algoritmos de recuperação através do processamento dos sinais digitalizados, o que "foge" do objetivo deste trabalho.

Já no caso dos arquivos MOD, além destes não constituírem um único padrão, a informação é parametrizada juntamente com a onda sonora representando a partitura, como visto no Capítulo 3. Nesse caso, a parte que representa a partitura é equivalente à sua representação em formato MIDI, o que levaria a um trabalho equivalente ao realizado, além do que, como não será manipulada a informação musical em nível de onda sonora, essa parte do formato ficaria sem tratamento.

Os padrões SBI e o IBK não representam músicas, mas instrumentos, o que os coloca fora do escopo deste trabalho também.

Desde abril de 1997, está sendo estudada a padronização de um novo formato de dados para partituras, que sejam adequados para sua comunicação na internet. Esse formato, denominado Rich

Music Format - RMF, deverá ser uma alternativa para o formato MIDI, e eventualmente poderá ser utilizado como um outro tipo de dado para a característica partitura [Aiki97].

6.1.2 Por que a Representação de Partitura e não Áudio Digitalizado ?

A representação de partituras pode ser feita de duas maneiras, como descrito na Seção 3.3: a representação por digitalização e a representação de áudio por síntese. Enquanto a representação digital tem a partitura em forma de onda sonora, a síntese tem a partitura parametrizada. Essa diferença não é adequada para que a partir da representação digital recupere-se uma partitura em partes, o que não ocorre com a recuperação sintetizada, a qual permite desde o controle das operações individuais de envelope até as mudanças de cada instrumento, como visto na Seção 3.3. A recuperação da informação digital seria através de algoritmos de processamento de sinais, o que foge da área descrita neste trabalho.

6.1.3 Remoção dos Meta-Eventos

No estudo referente ao formato MIDI, foi possível notar que este tipo de arquivo foi criado especificamente para tratar informações musicais, ou seja, foi demonstrada uma grande eficiência no tratamento de informações musicais, embora outras informações não musicais sejam também armazenadas junto, no mesmo arquivo MIDI, tais como nome de um instrumento (além de seu código), ou o título da música, armazenados como meta-eventos MIDI.

Os Sistemas Gerenciadores de Banco de Dados Orientados a Objetos- SGBDOO têm como função principal o gerenciamento de dados simples e não complexos como o áudio, ou partituras. No entanto, a informação representada pelos meta-eventos são de fato atributos simples, números e "strings" que podem ser adequadamente tratados pelos recursos usuais dos SGBDOOs. Por essa razão, passamos a "responsabilidade" de gerenciar esses dados (meta-eventos) para o SGBDOO, através da extração destes do arquivo MIDI. Enquanto o SGBDOO gerencia os dados, o formato MIDI apenas faz o tratamento de informações musicais.

Se fosse deixado que tanto o SGBDOO como o arquivo MIDI fizesse o gerenciamento de informações textuais, isso causaria inconsistência ou perda de tempo no gerenciamento dos dados, pois a atualização teria que ser feita em ambos.

6.1.4 Eliminação das Mensagens Exclusivas do Sistema

As mensagens exclusivas do sistema, como descritas na Seção 3.5.2, são informações não musicais exclusivas do fabricante. São informações de parametrização equivalentes às encontradas, por exemplo, no arquivo SBK, que indicam como deve ser feita a geração de um som. Por essa razão, apresentam problemas sérios de portabilidade. Além disso, as informações presentes nestas mensagens não deveriam ser guardadas como parte de um atributo partitura, e sim como definição de instrumento.

Como o banco de dados não possui tratamento para este tipo de atributo, ele simplesmente foi ignorado. Porém, pode ser feita como proposta de desenvolvimento futuro o suporte do tipo de dado (ou característica) Tipo de Instrumento Musical, permitindo que as mensagens exclusivas do sistema sejam tratadas pelo SGBDOO.

6.1.5 Por que MRO ?

No estudo realizado foram analisados vários modelos orientado a objetos, sendo dado enfoque a seus respectivos suportes a estruturas complexas, no caso objetivando o suporte a áudio. Dentre os modelos estudados, descritos no Capítulo 2, temos o Gem Stone, O2, Object Store e o MRO. Por esse estudo, pôde-se concluir que nenhum deles apresenta suporte a áudio. Assim o MRO é igualmente adequado a representação de áudio, ou seja, possui as mesmas facilidades básicas para representação de atributos não estruturados e longos. Como o grupo de banco de dados do ICMSC - Instituto de Ciências Matemáticas de São Carlos possui amplo conhecimento para efetuar alterações e extensões a esse modelo, o mesmo foi escolhido. No entanto, a menos da implementação final, todos os demais resultados obtidos são igualmente aplicáveis aos demais modelos.

6.2 Conclusões

Dentre os objetivos alcançados com o Editor de Partituras, temos a obtenção de um sistema de suporte à partitura uniforme e homogêneo, a armazenagem do MIDI dentro do banco de dados, e a extensão da linguagem do MRO-LAMRO.

6.2.1 Sistema de Suporte a Partitura Uniforme e Homogêneo

Conforme foi definido, o sistema de partituras manteve uniformidade e homogeneidade na representação de informações musicais. A uniformidade foi mantida pois trata a informação musical como mais um tipo de informação, ou seja, todas as características que um atributo em geral pode ter, o atributo partitura também pode. Apenas foi incluída funcionalidade, sem necessidade de alterações complexas e sem nenhum impacto na estrutura conceitual do modelo.

A homogeneidade foi mantida pelo fato de o usuário não ter muito a "aprender" para manipular partituras, já que o atributo partitura possui as mesmas características e é suportado pelo mesmo conjunto de comandos da linguagem de consulta já existente.

6.2.2 MIDI armazenado dentro do Banco de Dados

A recuperação no banco de dados pode ser feita de duas maneiras :

- armazenamento da partitura como um BLOB dentro do banco;
- armazenamento de um ponteiro para o arquivo partitura que é externo à base.

Na primeira opção, o armazenamento, recuperação, atualização, enfim todas as operações, são realizadas no banco de dados, permitindo que a mesma garanta a consistência de toda informação armazenada.

Na segunda opção, usando ponteiros para arquivo, a atualização não é feita a não ser que o banco de dados o faça, pois o gerenciamento é de responsabilidade do sistema operacional, que por sua vez não tem a função de atualizar o banco de dados. Isso abre a possibilidade de haver inconsistência a qual somente pode ser evitada pelos aplicativos e por um procedimento operacional sofisticado.

Através do armazenamento da partitura separada da informação textual do banco de dados, é possível exportar e extrair a informação, e ter a possibilidade de continuar usando as informações independente da sua característica. Isso não seria possível se o arquivo MIDI estive fora do banco de dados, pois a informação textual (meta-eventos) teria que ser mantida no arquivo MIDI, o que dificultaria sua manipulação.

6.2.3 MIDI dentro da Linguagem de Acesso Padrão LAMRO e SQL

Da mesma maneira que SQL possui uma estrutura algébrica representada pela álgebra relacional, temos a LAMRO como uma linguagem estrutural de acesso no caso do MRO. Como a representação

de partituras foi feita no MRO, então foi escolhida como linguagem padrão a LAMRO, já que esta implementa os conceitos suportados pelo modelo. Assim, tendo uma linguagem orientada a objeto é possível usá-la como apoio a uma linguagem de alto nível, por exemplo SQL3 (orientada a objeto) para o suporte a áudio.

A LAMRO, como descrita no Capítulo 4, foi estendida no que se refere aos comandos de acesso ao banco de dados, permitindo a importação, exportação, listagem, posicionamento, avanço e execução da partitura através dos comandos: `importa`, `exporta`, `lista`, `posiciona`, `avança` e `toca`, respectivamente.

6.2.4 Operações Usuais para Manipulação de Informações sobre Partituras em Banco de Dados

A criação de um novo tipo de dado em um modelo de dados implica a definição das operações de seleção e comparação. Este trabalho permitiu que fossem feitas as operações usuais de inclusão, comparação e remoção, envolvendo atributos com características musicais. Além do armazenamento de atributos do tipo partitura, foi dado o suporte a operações de seleção, comparação e remoção.

6.2.5 Recuperação por Conteúdo

O suporte a recuperação por conteúdo foi dado para permitir que se fizessem buscas de informações dentro da estrutura da partitura. Por exemplo, selecionar todas as partituras que tem no máximo 100 notas, implica a busca das informações existentes no arquivo MIDI.

6.3 Sugestões de Pesquisas Futuras

Esta seção propõe algumas linhas de pesquisas futuras que podem ser seguidas para complementar o Editor de Partituras e possivelmente incrementar melhorias.

6.3.1 Suporte a SQL

Assim como foi dado o suporte a LAMRO, é interessante estender a linguagem SQL3 permitindo a representação de atributos do tipo partitura.

6.3.2 Suporte a Operação de Alteração

O editor de partituras poderia ser complementado com funcionalidades de alteração, permitindo que fossem realizadas várias operações, entre elas:

- Transposição - permite que seja feita mudança no tom da partitura. Por exemplo, tocar uma partitura em dó maior ao invés de sol maior;
- Filtragem de Eventos - possibilita a remoção de parâmetros já existentes através da filtragem de alguns deles;
- Repetição - realiza a identificação ou geração da repetição resultando em acompanhamento automático, ou ao contrário, há possibilidade de identificar que há repetição e compactar a armazenagem.

A repetição pode ser associada a dois tipos de operações: **identificação (1a.operação)** que ocorre quando a repetição já existe na partitura e há necessidade de removê-la para economizar espaço de armazenagem. Para isso seleciona-se uma partitura que já tem, por exemplo, uma bateria e que possui o compasso homogêneo. Baseado nesta informação é necessária a **gravação (2a.operação)** do 1o. compasso e a indicação de que este se repetirá durante tantas vezes.

A identificação e gravação, por sua vez, pode resultar, por exemplo, na geração automática de acompanhamento ou de um trecho da música, bem como o suporte automático para:

- Arranjos - modifica o modo como a música é tocada pelos diversos instrumentos;
- Floreios - quando o autor "enfeita" a forma como o instrumento toca sua parte.

6.3.3 Análise de Partituras

A análise de partituras envolve o desenvolvimento de algoritmos para levantamento das informações referentes a partituras. Por exemplo, é possível selecionar uma coleção de partituras que sejam do mesmo autor. Isso é feito para saber se este segue algum padrão em suas composições, para a criação de um padrão. Do ponto de vista do banco de dados, isso corresponde à extração de informação estatística, permitindo realizar, por exemplo, operações do *data-mining* pelo conteúdo de partituras.

6.3.4 Geração de Trechos ou Trilhas

A geração de trechos ou trilhas é baseada em regras existentes ou algumas vezes, em regras

descobertas através da análise de partituras. Por exemplo, existem estudos que permitem identificar uma espécie de "assinatura" musical que identifica o autor de uma peça musical. Partindo dessas informações pode-se aplicar regras existentes para gerar música aleatoriamente, que no entanto refere-se a forma como um autor escreveria.

Na **geração de trechos** há a seleção de um conjunto de trilhas em um determinado intervalo de tempo, enquanto na **geração de trilhas** uma trilha é selecionada para ser manipulada, permitindo a geração automática de acompanhamento ou de floreio. Por exemplo, considerando que haja um violino como solo e um violoncelo fazendo o acompanhamento, e foi necessário fazer uma alteração nos acordes. Neste caso, houve geração de trechos pois foi feita alteração em algo já existente.

Com a informação guardada toda separada no banco de dados, é possível a aplicação das regras existentes, facilitando a codificação de programas para manipulação das partituras. Logo, basta fazer programas que acessam o banco de dados como aplicativos para analisar e modificar as partituras existentes.

Bibliografia

Bibliografia

[Aiki97] Aikin, J.- **"MIDI Meests the Internet"**, Keyboard - vol.23, #7, p.36-51, July 1997.

[Banc92] Bancilhon, F.- **"The O_2 Object-Oriented Database System"**, Proceedings of the 1992 ACM Sigmod - International Conference of Management of Data, San Diego, California, vol.2, No.1, pp.7, June 1992.

[Bark95] Barkakati, N.- **"Imagens e Animação Gráficas para Windows"**, Capítulo 5, pp.111-127, Editora Ciências Modernas, 1995.

[Bert91] Bertino, E.; Martino, L.- **"Object Oriented Database Management Systems : Concepts and Issues"**, IEEE Computer Society, 24(4), pp.33-47, 1991.

[Bert93] Bertino, E.; Lorenzo, M.- **"Object-Oriented Database Systems"**,International Computer Science Series, Addison-Wesley, 1993.

[Biaj92] Biajiz, M.- **"MRO* - Uma Atualização do Modelo de Representação de Objetos e uma Abordagem Formal"**- Dissertação de Mestrado- Universidade de São Paulo, Instituto de Ciências Matemáticas de São Carlos, 1992.

[Butt91] Butterworth, P.; Otis,A.; Stein, J.- **"The GemStone Object Database Management System"**, Communications of the ACM, vol.34, no.10, pp.64-77, October 1991.

[Camo92] Camolesi, L.- **"Suporte a Acesso Multi-Usuário em Base de Dados Orientada a Objetos através de Esquemas Suplementares"** -Dissertação de Mestrado- Universidade de São Paulo, Instituto de Ciências Matemáticas de São Carlos, 1992.

[Catt94] Cattell, R.G.G. - **"Object Data Management"**, Addison-Wesley Publishing Company, 1994.

[Darw95] Darwen. H.; Date, C.J- **"The Third Manifesto"**, Sigmod Record, vol.24, No.1, March 1995.

[Deux91]- Deux, O. et al.; **"The O_2 System"**, Communications of the ACM, vol.34, no.10, pp.34-48, October 1991.

[Figu96]- Figueiredo, M.B; Júnior, C.T.- **"GEO-Gerenciador de Objetos"** - Relatório Técnico No.47- Universidade de São Paulo, Instituto de Ciências Matemáticas de São Carlos, 1996.

[Gome93] Gomes, T.A.; Neves, A.- **"Tecnologia Aplicada à Música"**, Editora Érica, 1993.

[Hube95] Huber, M.D.- **"The MIDI Manual"**, SAMS, Prentice Hall Computer Publishing, 1995.

[Jose91] Joseph, J.; Thatte, S; Thompson, C.; Wells, D.- **"Object Oriented Databases: Design and Implementation"**, IEEE Proceedings, 79(1), pp.42-64, 1991.

[Kim89] Kim, W.; Lochovsky, F.- **"Object Oriented Concepts, Databases and Applications"**, Reading MA: Addison-Wesley, 1989.

[Kim90] Kim, W.- **"Introduction to Object Oriented Database"**. Cambridge MA: The MIT Press, 1990.

[Lamb91] Lamb, C.; Landis, G.; Oreinstein, J.; Weinreb, D.- **"The ObjectStore Database System"**, Communications of the ACM, vol.34, No.10, pp.50-63, October 1991.

[Maie89] Maier, D.; Zdonik, S.- **"Fundamentals of Object Oriented Database Management Systems"**, Morgan Kauffman, 1989.

[Ming95] Minghim, R. - **"On Sound Support for Visualisation"**, School of Information Systems - University of East Anglia, 1995.

[Obje95] Object Design, Inc.- **"Object Store Technical Overview"** *Uniform Resource Locators (URL).* [online] 1995. Avaiable from World Wide Web:<URL:http://www.odi.com/products/os/techovrwv.html> [Fev.1996].

[Pent90] Pentfold, R.A. - **"Practical MIDI Handbook"**, PC Publishing, 1990.

[Corr92] Correa, P.L.P.- **"Uma Linguagem para o Gerenciador de Objetos Baseada no Modelo de Representação de Objetos"** -Dissertação de Mestrado- Universidade de São Paulo, Instituto de Ciências Matemáticas de São Carlos, 1992.

[Ratt95] Ratton, M.- **"Criação de Música e Sons no Computador"**, Editora Campus, 1995.

[Serv87] Servio's GemStone- **Object-Orientation FAQ-GemStone (Servio)"** *Uniform Resource Locators (URL).* [online] 1987. Avaiable from World Wide Web:<URL:http://www.ipipan.gda.pl/~marek/objects/faq/00-faq-s-8.12.1.3.html> [Fev.1996].

[Stol93] Stolz, A.- **"The Sound Blaster Book"**, Abacus, 1993.

[Trai92] Traina, C.T.; Slaets, J.F.W.- **"MRO-Um Modelo de Representação de Objetos"** - Notas do ICMSC -Universidade de São Paulo, Instituto de Ciências Matemáticas de São Carlos, 1992.

[Voss91] Vossen, G.- **"Data Models, Database Languages and Database Management Systems"**, Reading MA: Addison-Wesley, 1991.

[Zand95] Zand, M.; Collins, V.; Caviness, D.- **"A Survey of Current Object-Oriented Databases"**, Database Advances, vol.26, No.34, pp.14-29, February 1995.

[Zhan95] Zhang, A.- **"Multimedia File Formats on the Internet"**, *Uniform Resource Locators (URL)*. [online] 1995. Avaiable from World Wide Web:<URL:http://rodent.lib.rochester.edu/multimed/contents.htm> [Fev.1996].

[Woel87] Woelk, D.; Kim, W.- **"Multimedia Information Management in an Object-Oriented Database System"**, In Proc.of the International Conference on Vey Large Databases (VLDB), Brighton, UK, September 1987.

Bibliografia Complementar

[Bagg91] Baggi, D.- **"Guest Editor's Introduction :Computer-Generated Music"**, IEEE Computer Society, vol.24, No.7, pp.6-11, July 1991.

[Baro81] Baroody, A.J.; DeWitt, D.J. - **"An Object-Oriented Approach to Database System Implementation"** - ACM Transactions on Database Systems, vol.6, No.4, pp.577-601, December 1981.

[Borr92] Borras, P.; Mamou, J.C.; Poyet, B.; Tallot, D.- **"Building user interfaces for database applications: The O_2 experience"**, Sigmod Record, vol.21, No.1, pp.32-38, March 1992.

[Doga93] Dogac, A.; Arpinar, B.- **"METU Object-Oriented DBMS"**, pp.513, 1993.

[Hull89] Hull, R.; Su, J.- **"On Acessing Object-Oriented Database: Expressive Power, Complexity, and Restrictions"**, ACM, pp.147-158, 1989.

[Ishi93] I., Hiroshi; Suzuki,F; Kozakura, F.- **"The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System"**, ACM Transactions on Database Systems, vol.18, No.1, Março 1993.

[Kife92] Kifer, M.; Kim, W.; Sagiv. Y.- **"Querying Object-Oriented Databases"**, ACM Sigmod, pp.393-402, 1992.

[Kim93] Kim, W.; Suh, Y.; Whinston, A.B.- **"An IBIS and Object-Oriented Approach to Scientific Research Data Management"** , J.Systems *Software*,vol. 23, pp.183-197, 1993

[Madh95] Madhyastha, T.M; Reed, D.A.- **"Data Sonification: Do you see what I hear?"**, IEEE *Software*, pp.45-56, March 1995.

[Mari92] Mariani, J.A.- **"Oggetto: An Object Oriented Database Layered on a Triple Store"** - The Computer Journal, vol.35, No.2, pp.108-118, 1992.

[Milb93] Milburn, K.- **"Sound Off ! Sound Cards let you 'talk up' your data"**, Windows Magazine Shopper, vol.4, No.8, pp.226-260, August 1993.

[Moor90] Moore, D.J.- **"Multimedia Presentation Development using the Audio Visual Connection"**, IBM Systems Journal, vol.29, No.4, pp.494-508, 1990.

[Nava92] Navathe, S.B. - **"Evolution of Data Modeling for Databases"**, Communications of the ACM, vol.35, No.9, pp.112-123, September 1992.

[Reic92] Reichbach, J.D.; Kemmerer, R.A.- **"Sound Works: An Object-Oriented Distribuited System for Digital Sound"**, IEEE Computer Society , vol.25, no.3, pp. 25-37, Março 1992.

[Vask92] Vaskevitch, D.- **"Two Steps Forward, One Step Back"** - Byte, pp. 141-146, May 1992.

[Vele93] Vélez, F.- **"Modularity and Tuning Mechanisms in the O_2 System"**, ACM Sigmod, pp.440, 1993.

[Zhao88] Zhao, L.; Roberts, S.A.- **"An Object-Oriented Data Model for Database Modelling Implementation and Access"** - The Computer Journal, vol.31, No.2, pp.116-124,1988.

[Well92] Wells, D.L.; Blakeley, J.A.; Thompson, C.W.- **"Architecture of an Open Object-Oriented Database Management System"** - IEEE Computer, pp.74-81, October 1992.

Apêndice

Baseado nos comandos demonstrados e no exemplo abaixo, usaremos a LAMRO para demonstrar o esquema de dados e a base de dados de um sistema.

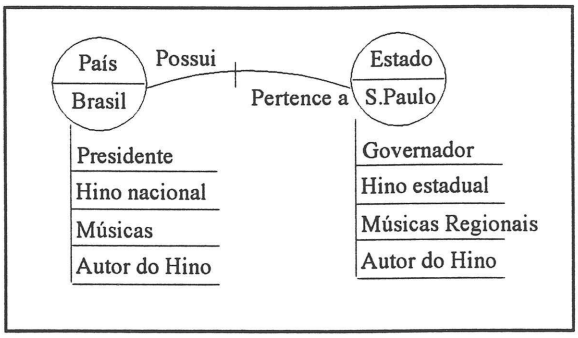

Figura 1- Esquema de Dados

Esquema de Dados

```
liga meta;                                          /* liga a colônia Besquema */
objeto objeto pais, estado;                         /* cria objetos do tipo País e Estado */
localiza colonia global;                            /* posiciona na colônia Global */
relacionamento habitada_por objeto pais,estado;     /* define os objetos que habitam a colônia
                                                    Global */
objeto modalidade1 possui, pertence_a;              /* define as modalidades */
localiza modalidade1 possui;                        /* posiciona na modalidade Possui */
        relacionamento oposto modalidade1 pertence_a; /* define a modalidade oposta a Possui */
objeto binario R1;                                  /* define o 1o.relacionamento binário */
        relacionamento composto modalidade1 possui;     /* define a composição do
                                                        relacionamento */
        relacionamento origem objeto pais;          /* define os objetos pertencentes ao
                                                    relacionamento */
        relacionamento destino objeto estado;       /* define os objetos pertencentes ao
                                                    relacionamento */
objeto binario R2;                                  /* define o 2o.relacionamento binário */
        relacionamento composto modalidade1 pertence_a; /* idem definição do relacionamento
                                                        anterior */
        relacionamento origem objeto estado;
```

```
        relacionamento destino objeto pais;
objeto atributo presidente, hino nacional, musicas, autor do hino; /* define os tipos de atributos
do                                                                  objeto país */
localiza atributo presidente;                          /* posiciona no atributo presidente */
        propriedade tipo_de_dado 7;                    /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica 2;                  /* define o atributo como tendo a
                                                       característica de propriedade */
localiza atributo hino nacional;                       /* posiciona no atributo hino nacional */
        propriedade tipo_de_dado 12 (BLOB);            /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica 12 (PARTITURA);
localiza atributo musicas;
        propriedade tipo_de_dado ;                     /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica ;                   /* define o atributo como tendo a
                                                       característica de propriedade */
localiza atributo autor do hino;
        propriedade tipo_de_dado ;                     /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica ;                   /* define o atributo como tendo a
                                                       característica de propriedade */
localiza objeto estado;
objeto atributo governador, hino estadual, musicas regionais, autor do hino;
localiza atributo governador;
        propriedade tipo_de_dado ;                     /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica ;                   /* define o atributo como tendo a
                                                       característica de propriedade */
localiza atributo hino estadual;
        propriedade tipo_de_dado ;                     /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica ;                   /* define o atributo como tendo a
                                                       característica de propriedade */
localiza atributo musicas regionais;
        propriedade tipo_de_dado ;                     /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica ;                   /* define o atributo como tendo a
                                                       característica de propriedade */
localiza atributo autor do hino;
        propriedade tipo_de_dado ;                     /* define o atributo como um tipo unsigned
                                                       char */
        propriedade caracteristica ;                   /* define o atributo como tendo a
```

característica de propriedade */
mostra Esquema;
desliga meta;

Base de Dados
objeto pais brasil, argentina, espanha, italia, australia;
localiza pais brasil;
propriedade presidente "Fernando H.Cardoso";
propriedade hino nacional brasil.mid;
propriedade musicas mpb.mid;
propriedade autor do hino "Francisco ....."
localiza pais argentina;
propriedade presidente;
propriedade hino nacional argentina.mid;
propriedade musicas tango.mid;
propriedade autor do hino;
localiza pais espanha;
propriedade presidente;
propriedade hino nacional espanha.mid;
propriedade musicas flamenga.mid;
propriedade autor do hino;
localiza pais italia;
propriedade presidente;
propriedade hino nacional italia.mid;
propriedade musicas massa.mid;
propriedade autor do hino;
localiza pais australia;
propriedade presidente;
propriedade hino nacional australia.mid;
propriedade musicas canguru.mid;
propriedade autor do hino;
localiza pais brasil;
objeto estado sao paulo, rio grande do sul, minas gerais;
localiza estado sao paulo;
propriedade governador "Mario Covas";
propriedade hino estadual hino_sp.mid;
propriedade musicas regionais garoa.mid;
propriedade autor do hino;
localiza estado rio grande do sul;
propriedade governador "";
propriedade hino estadual hino_rs.mid;
propriedade musicas regionais gaucho da fronteira.mid;
propriedade autor do hino;

localiza estado minas gerais;
propriedade governador "";
partitura hino estadual hino_mg.mid;
propriedade musicas regionais milton nascimento.mid;
propriedade autor do hino;

De acordo com estes dados, se quisermos tocar o hino estadual de São Paulo e juntamente saber o autor do hino pertencentes ao estado basta seguir os comandos :

posiciona hino_sp.mid;
liga partitura [autor do hino];
lista;
toca hino_sp.mid;