O Protocolo MMS: Projeto e Aplicações

Frederick Bigoni Burrowes
Instituto de Pesquisas da Marinha
Rua Ipiru, 2
Ilha do Governador - Rio de Janeiro - RJ

Márcia Gonçalves de Novaes
Mira Informática Ltda.
Av. Pres. Wilson, 164 - 2° andar
20030 - Rio de Janeiro - RJ

Aloysio de Castro Pinto Pedroza
EE e COPPE/UFRJ - Programa de Engenharia Elétrica
Centro de Tecnologia - H-321 - CP 68504
Cidade Universitária
21945 - Rio de Janeiro - RJ

Resumo

Este trabalho apresenta o projeto e a implementação de um protótipo do protocolo de comunicação de dados MMS (Message Manufacturing Specification) para uma arquitetura Mini-MAP. O projeto e a implementação do protocolo MMS foram desenvolvidos usando uma metodologia orientada a objetos. Também são apresentados dois tipos de aplicação: um sistema naval e um sistema de coleta de dados fabril.

Abstract

This paper presents the project and a prototype implementation of the MMS (Message Manufacturing Specification) data communication protocol for a Mini-MAP architecture. The design and the implementation of the MMS protocol were developed using an object-oriented methodology. Two examples of a possible use of the MMS protocol are also presented: a naval system and an industrial data collection system.

I. Introdução

Este trabalho apresenta o projeto e a implementação orientados a objeto de um protótipo do protocolo de comunicação MMS (Message Manufacturing Specification) [ISO87]. Os resultados aqui descritos são oriundos de trabalhos efetuados com o intuito de avaliar o referido protocolo [BUR92] [NOV92]. O projeto e a implementação efetuados tiveram como alvo uma arquitetura Mini-MAP [GMM87].

Descreve-se também dois tipos de aplicações. A primeira corresponde ao uso do protocolo MMS em um sistema de coleta de dados fabril. A segunda corresponde a um exemplo da viabilidade de uso do protocolo MMS em aplicações navais.

As observações desenvolvidas são exemplos da aplicabilidade do uso da orientação a objetos em sistemas de tempo real, em particular no que se refere a protocolos de comunicação.

A seção II deste trabalho descreve a arquitetura Mini-MAP. A seção III apresenta o conceito de interfaces de aplicação. Discute-se a modelagem do sistema na seção IV. Na seção V descreve-se o projeto orientado a objetos do protocolo MMS. As razões que levaram à escolha de C++ como linguagem de implementação são relacionadas na seção VI. O relacionamento entre o projeto orientado a objetos e o mapeamento sobre o sistema operacional é objeto de discussão na seção VII. Os testes efetuados são apresentados na seção VIII. Por último, os exemplos de aplicação, na seção IX.

II. A Arquitetura Mini-Map

Em 1979, foi iniciado na General Motors o projeto MAP [GMM87], Manufacturing Automation Protocol, visando a especificação de protocolos apropriados para a automação industrial. Dentro do projeto MAP, a opção Mini-MAP surge como alternativa para ambientes tempo real com tempos de resposta entre 5 e 100 ms e mensagens curtas, de 50 a 250 bytes.

Basicamente, a arquitetura Mini-MAP elimina quatro das sete camadas do ambiente OSI/ISO (Open System Interconnection/International Organization for Standardzation), ficando apenas com: a física, a de enlace e a de aplicação. Desta forma consegue-se um substancial aumento nos tempos de troca de mensagens.

O uso de apenas três camadas na opção Mini-MAP acarreta a perda dos serviços das camadas de rede, transporte, sessão e apresentação. A opção Mini-MAP especifica na máquina de protocolos, a MMPM (Message Manufacturing Protocol Machine), mecanismos que contornam parte da perda destes serviços, principalmente na área da confiabilidade da entrega de mensagens. Além destas três camadas, a literatura [MEN89] [FON85] tem apontado a necessidade de uma interface denominada de API (Application Program Interface), atuando como uma camada que não gera PDU's. Esta necessidade não é privilégio de arquiteturas MAP, mas de arquiteturas OSI/ISO em geral.

A camada Física especificada para o Mini-MAP segue as opções baseadas na norma IEEE 802.4 [IEE88b]. Já a camada de enlace utiliza o protocolo IEEE 802.2, tipo 1 ou 3. Com relação à camada de aplicação, numa rede MAP, bem como numa rede Mini-MAP, podem coexistir diversos protocolos ISO de aplicação. Entretanto, sem dúvida, o principal destes é o MMS [ISO87]. A implantação da camada de aplicação do Mini-MAP é feita através da MMPM. Em verdade, no que se refere à MMPM, podemos afirmar que esta é mais complexa que a MMPM do FULL-MAP, por ser responsável pela realização de funções tanto da subcamada ACSE, inexistente na opção Mini-MAP, quanto do mapeamento direto no nível 2.

A figura I ilustra o relacionamento entre a MMPM, a camada de enlace e a API, sendo esta última descrita a seguir. Inicialmente o programa de aplicação do usuário solicita um serviço à API através de uma chamada de procedimento de uma biblioteca. A API por sua vez realiza um "request" à máquina de protocolos. Esta trata a primitiva e envia para o nível 2 uma primitiva L_Data.request que passa para a camada inferior. Quando chega a mensagem no nível 2 da estação remota, vinda do nível 1 desta, é enviada uma primitiva L_data.indication para a máquina de protocolos remota que então responde para a API. Esta, após realizar o serviço solicitado, envia uma primitiva do tipo "response", que, fazendo o caminho de volta, chega ao usuário que iniciou a comunicação como uma primitiva "confirm".

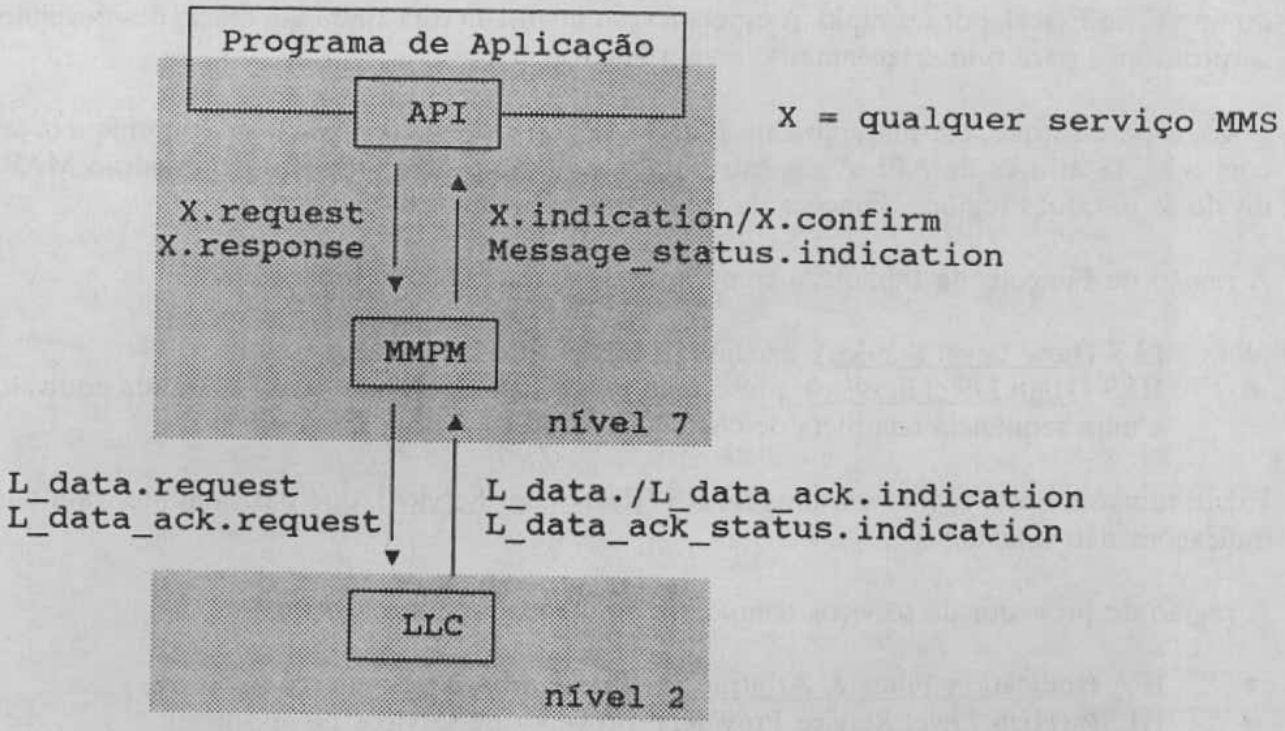


Figura I. Relacionamento entre API, MMPM e LLC

III. A Interface de Aplicação

A Interface de Aplicação, API (Application Program Interface), oferece mecanismos de acesso aos Application Service Elements (ASEs), de maneira que um usuário não tenha que acessar diretamente os protocolos de aplicação, como o MMS, por exemplo. Cada ASE engloba um conjunto de funções para a realização de tarefas específicas de comunicação.

O uso de uma API possibilita as seguintes vantagens:

- Separa o usuário dos complexos mecanismos em geral presentes em protocolos de aplicação. Desta forma, uma API bem especificada pode facilitar em muito a tarefa de programação;
- Facilita a portabilidade de programas.

Atualmente, em âmbito internacional, existe um intenso esforço no sentido de padronização de APIs, tendo em vista o fator portabilidade de software aplicativo que tal padronização acarretará.

No que se refere ao projeto MAP, existe uma sugestão de API, a qual foi especificada, na medida do possível, de modo a ser independente de sistemas operacionais. Desta forma, a única exigência que a especificação da API do grupo MAP estabelece quanto ao sistema operacional é que este seja capaz de ativar e desativar processos [GMM88].

A especificação da API está escrita de forma genérica tendo em vista linguagens procedurais do tipo C ou Pascal, por exemplo. A especificação analisada traz ainda um anexo descrevendo os protótipos para o interfaceamento com a linguagem C.

Logo, espera-se que, em um ambiente Mini-MAP, os programas aplicativos comuniquem-se com o MMS através da API. A sua estrutura interna funcional, proposta pelo padrão MAP, divide-se em duas regiões: Funções da Biblioteca e Provedor de Serviços.

A região de Funções da Biblioteca compõe-se de duas classes de bibliotecas:

- LLS (Low Level Service): executa os serviços de baixo nível;
- HLS (High Level Service): prevê uma maior funcionalidade - uma chamada equivale a uma sequência completa de chamadas MMS.

Existe também nesta região um módulo RS (Responder Service), que basicamente trata das indicações não solicitadas.

A região de provedor de serviços compõe-se de quatro módulos funcionais:

- IFA (Indication Filter & Arbitrator): filtro e árbitro de indicações;
- HLSP (High Level Service Provider): provedor de serviços de alto nível;
- CSP (Confirmed Service Provider): provedor de serviços confirmados;
- PSP (Primitive Service Provider): provedor de serviços primitivos.

IV. A Modelagem do Sistema

Com o intuito de se obter um melhor conhecimento do MMS, dentro do escopo de [BUR92] e [NOV92], procedeu-se ao projeto de um protótipo de MMS, simulando-se uma arquitetura Mini-MAP. Tal projeto constitui-se das fases de especificação formal, projeto propriamente dito e implementação.

A especificação formal da MMPM foi efetuada em linguagem Estelle, tendo sido compilada com o auxílio de um CAD de protocolos [PED89]. Esta especificação deu origem à arquitetura formal do sistema, a qual encontra-se representada na figura II [BUR89].

Na fase da especificação da API decidimos pela utilização da técnica de orientação a objetos. A modelagem com objetos surgiu naturalmente, tendo em vista que a norma MMS [ISO87]

usa uma linguagem totalmente voltado para o conceito de objetos.

Entretanto, a adoção da orientação para objetos apresentava alguns problemas. Encontra-se tradicionalmente estabelecido pela Engenharia de Software que um projeto de um sistema engloba pelo menos três fases: Análise (especificação), Projeto e Implementação. Um projeto totalmente orientado a objetos englobaria também estas três fases. A fase de análise, que vem se tornando conhecida em publicações internacionais de Engenharia de Software como Object Oriented Analysis (OOA) [SHL88], a fase de projeto propriamente dito, conhecida como Object Oriented Design (OOP) [COA91], e, finalmente, a fase de implementação onde encontramos as linguagens orientadas a objetos ou Object Oriented Program Languages (OOPL). Ocorre, entretanto, que as duas primeiras fases são relativamente novas, não existindo, atualmente, metodologia solidificada para elas. O que existe sim são as diversas linguagens orientadas a objetos.

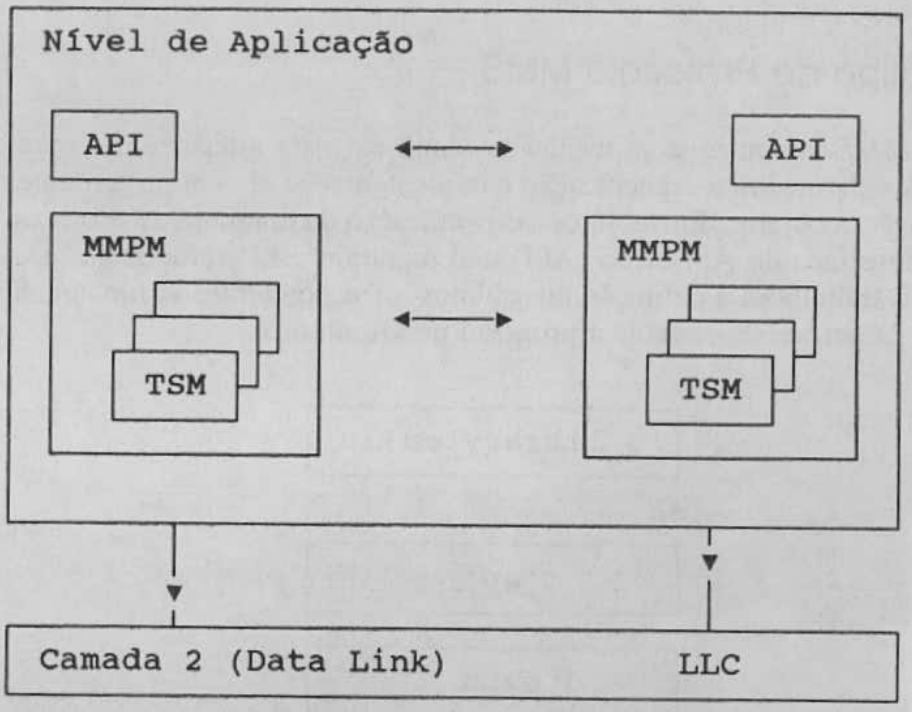


Figura II. Arquitetura da Modelagem em Estelle

No que se refere à especificação de protocolos, mesmo estas propostas se revelam pouco adequadas pois não possuem o formalismo de métodos como Estelle e Lotos, por exemplo. A solução adotada foi especificar a MMPM em Estelle e considerar a especificação da API [GMM88] como a especificação do projeto, na falta de melhor metologia. Na realidade esta última citada especificação apresenta muitos aspectos típicos de projeto, como é o caso dos bindings em linguagem C.

Para a fase de projeto o mesmo problema se apresentou. A solução foi usar representações gráficas que pareceram apropriadas à época. Meses depois constatou-se que as idéias usadas vinham de encontro com algumas das primeiras propostas na área [COA91] [DEA91]. O que basicamente se procura nestas metodologias é a representação dos objetos e classes, com seus

atributos, representando-se também a interação entre estes objetos.

Assim, o desenvolvimento do protótipo compreendeu as seguintes etapas:

- modelagem da MMPM em Estelle;
- definição dos objetos que compõem o sistema;
- mapeamento destes objetos no sistema operacional escolhido;
- implementação em C++, em microcomputadores compatíveis IBM-PC.

A implementação resultante teve como objetivo principal auxiliar no estudo do protocolo MMS, dentro do conceito de prototipagem evolutiva. Neste tipo de prototipagem, primeiramente desenvolve-se uma versão preliminar, o protótipo. A partir dos conhecimentos obtidos com o protótipo pode-se evoluir com o sistema.

V. Protótipo do Protocolo MMS

A norma MMS encontra-se especificada tendo em vista a técnica de orientação a objetos, o que sem dúvida facilita a especificação e implementação em um projeto que use metodologias de orientação a objetos. Entretanto, a especificação da máquina de protocolos (MMPM) bem como da Interface de Aplicação (API) não seguiram esta metodologia. Desta forma, foi um pouco mais trabalhosa a definição dos objetos correspondentes às funcionalidades da MMPM e da API. Descreve-se a seguir a proposta de arquitetura.

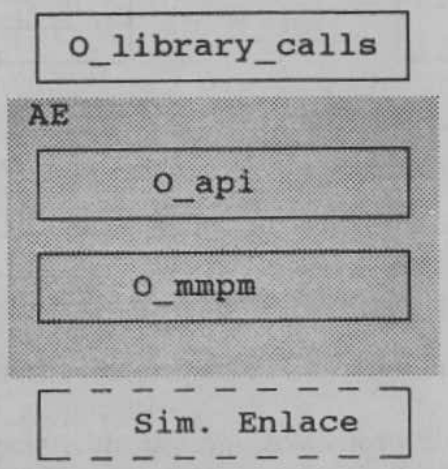


Figura III. Mapeamento em Classes de Objetos

A figura III representa o mapeamento em classes de objetos usados no protótipo abstraindo-se das considerações quanto ao mapeamento no sistema operacional. Somente os principais objetos encontram-se representados nesta figura, de forma que objetos herdados ou internos a estes objetos representados são evitados como forma de ser possível uma abordagem top-down, de acordo com as idéias apresentadas em [DEA91]. Na citada figura, a classe O_Library_calls modela a entidade de mesmo nome, a menos do "O_", definida na especificação de API do grupo MAP [GMM88]. O conjunto de classes compostas por O_api

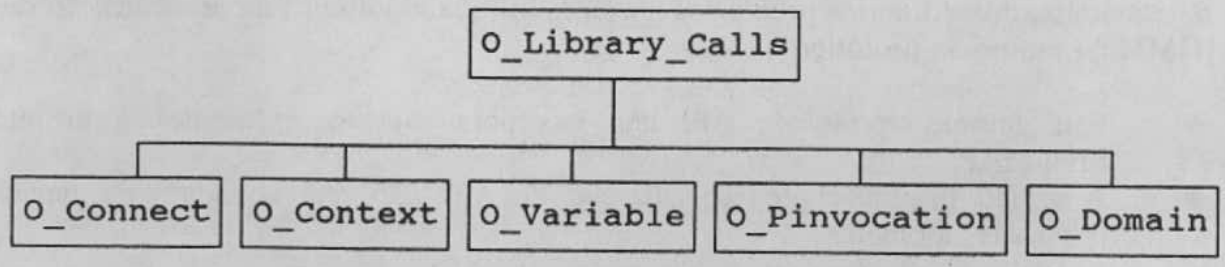
e O_mmpm modelam a entidade conhecida como Application Entity (AE) no modelo de referência OSI/ISO. A entidade AE engloba as facilidades de comunicação de um Application Process (AP).

Ainda na figura III, a classe O_api modela as funcionalidades da API conforme descrito na especificação de API do grupo MAP. A classe O_mmpm modela as funcionalidades da máquina de protocolos, definida pelo grupo MAP para arquiteturas Mini-MAP. Como vemos, os objetos citados procuram retratar entidades definidas nas normas utilizadas. Por fim, existe ainda um conjunto de procedimentos que implementam um simulador de enlace, o qual encontra-se descrito em [NOV92].

O item V.1 a seguir apresenta a classe O_Library_calls, com as modificações propostas nesta classe em relação à especificação original, o item V.2 a classe O_api e finalmente, o item V.3 a classe O_mmpm.

V.1. A Classe O_Library_calls

Na figura IV a classe chamada de O_Library_calls corresponde as funções do bloco Library Calls encontrado na especificação da API parte superior. Conforme as funções usadas na implementação, as quais são determinadas pelo Companion Standard, tal objeto herdará características de outros objetos, dentro da filosofia de inheritance da metodologia de orientação a objetos. Esta classe responsabiliza-se por fornecer ao usuário os métodos a serem usados para interfaceamento com os AE's (Application Entities).



aonde: O_Connect = O_Connection_manegement_library_calls
O_Context = O_Context_free_library_calls
O_Variable = O_Variable_access_service_library_calls
O_Pinvocation = O_Program_invocation_service_library_calls
O_Domain = O_Domain_management_service_library_calls

Figura IV. A classe O_Library_Calls

A classe O_Library_calls sofreu alterações em relação ao documento de especificação utilizado [GMM88]. Estas modificações, bem como um detalhamento maior desta classe, estão na seção a seguir.

A figura IV representa a classe O_Library_calls com as classes herdadas por esta em nosso protótipo. O classe O_Connection_manegement_library_calls é a classe encarregada das funções de apoio a conexão. Já a classe O_Context_free_library_calls corresponde às funções

de apoio independentes de contexto. Finalmente, a classe O_Variable_access_library_calls corresponde às funções de gerência de variáveis. Estas classes herdadas correspondem a blocos funcionais de mesmo nome (a menos do "O_"), encontrados na especificação de API. Caso novas funcionalidades não implementadas se façam necessárias, novas classes de objetos, correspondentes a estas funcionalidades, podem passar a ser herdados por O_Library_calls.

Tendo-se optado pela forma de comunicação sem associação, como forma de reduzirmos o overhead, as funções da API implementadas foram:

- Serviços comuns: Application Entity Activation, Application Entity Deactivation, Indication Receive, Cancel e Cancel Response;
- Serviço de Acesso a Variáveis: New Variable Association, Make List, Define Local Variable, Delete Local Variable, Define Local List, Local List Delete, Read e Report Information;
- Serviço de Invocação de Programa: Create Program Invocation, Delete Program Invocation, Start, Stop, Resume, Reset e Get Program Invocation Attributes;
- Serviço de Gerenciamento de Domínios: Load Domain Content e Delete Domain;

V.1.1. Proposta para API do Mini-MAP

As modificações propostas para a especificação da API-MMS visando a orientação a objetos e a arquitetura Mini-MAP apresentam estreito relacionamento com a classe de objeto O_Library_Calls.

Basicamente, dois foram os problemas encontrados quando do uso da especificação de API [GMM88] em nosso protótipo:

- Esta mesma versão de API não incorpora aspectos referentes à arquitetura Mini-MAP;
- A versão disponível de especificação de API não está voltada para linguagens orientadas a objetos.

A solução do primeiro problema foi relativamente fácil. Basicamente foram incluídos nas funções da API parâmetros correspondentes à prioridade de comunicação e ao endereçamento. Passamos agora a tratar dos aspectos referentes ao segundo dos tópicos acima listados.

Conforme encontramos textualmente explicado no draft utilizado da especificação de API, tal especificação encontra-se voltada para linguagens procedurais, do tipo Pascal ou C, estando indeterminada a aplicabilidade desta em linguagens orientadas a objetos. Desta forma, tendo-se em vista a decisão de adotarmos uma implementação voltada à orientação a objetos, duas opções foram consideradas no que se refere à implementação da API. A primeira seria manter o interfaceamento com o usuário conforme encontra-se descrito na citada especificação de API, realizando a implementação orientada a objeto apenas nos aspectos internos da API. Ou seja, na figura III a classe denominada de O_Library_calls não seria uma classe - estaria implementada baseada em linguagens procedurais e os demais módulos seguiriam a orientação a objetos. Esta primeira opção não demandaria praticamente nenhum

afastamento em relação à especificação de API, já que a mesma, a menos de questões referentes ao interfaceamento com o usuário, não especifica como mas o que implementar.

Uma segunda opção, tendo esta sido a escolhida, era implementar a API totalmente dentro da filosofia de orientação a objetos. Um estudo detalhado da especificação de API revelou que esta poderia ser adaptada para a filosofia da orientação a objetos, bastando para tanto pequenas modificações. Apresenta-se então as modificações propostas e utilizadas no protótipo, lembrando antes que estas são apenas uma primeira sugestão, sendo possível, e até esperado, que no futuro mecanismos melhores venham a ser apresentados. São duas as principais características dessa proposta:

- A especificação de API [GMM88] continua válida. Na hipótese da adoção da orientação a objetos, bastaria um pequeno anexo para torná-la compatível com tal filosofia;
- O uso da orientação a objetos torna o uso da API mais fácil para o usuário, já que facilidades típicas, encontradas na orientação a objetos, podem vir a ser incorporadas.

A classe O_api segue a descrição prevista na citada especificação para a chamada área de Interface Service Provider (Provedor de Serviços de Rede). As citadas sugestões de modificação estão concentradas na classe O_Library_calls. A seguir são detalhadas explicitamente.

A classe O_Library_calls corresponde àquela região denominada de Funções da Biblioteca na especificação da API. Como podemos observar na figura IV, esta classe herda uma série de outras classes de objetos, todos com mapeamento direto em módulos definidos na especificação de API. Todos estes módulos possuem funções, acessíveis ao usuário através de Function Calls (isto se estivermos pensando no binding para a linguagem C). A sugestão de implementação é que estas funções passem a corresponder a metódos dos referidos objetos. Assim, resumidamente, a declaração em pseudo C++ da classe de objeto Variable_access_service_calls seria a seguinte:

```
Class Variable_acess_service_library_calls:  Interface
{
        public:
                Return_code  mm_dlvariable (O_mm_dlvariable& param);
                Return_code  mm_dvdelete  (O_mm_dvdelete& param);
                Return_code  mm_dllist      (O_mm_dllist& param);
                Return_code  mm_lldelete   (O_mm_lldelete& param);
                Return_code  mm_read       (O_mm_read& param);
                Return_code  mm_write      (O_mm_write& param);
}
```

Como já mencionamos cada método desta classe corresponde a uma função encontrada na especificação da API. A título de exemplo segue-se o binding em C correspondente a dois métodos, conforme podemos encontrar na especificação:

```
/* dlvariable */
```

```
Return_code  mm_dlvariable (connection_id, return_event_name, input_dcb, inout_dcb)
Mm_Vmd_id                  connection_id;
Local_event_name           local_event_name;
Mm_dlvariable_in_dcb       *input_dcb;
Mm_dlvariable_out_dcb      **inout_dcb;
{     /* empty function body */     }

/* Read */
Return_code  mm_read (connection_id, return_event_name, input_dcb, inout_dcb)
Connection_id              connection_id;
Local_event_name           local_event_name;
Mm_Read_in_dcb             *input_dcb;
Mm_Read_out_dcb            **inout_dcb;
{     /* empty function body */     }
```

Portanto, os métodos correspondem às funções. Entretanto, como pode-se observar, os parâmetros passados a estes métodos apresentam-se de forma bastante diferente daqueles encontrados nas funções reproduzidas do binding em C. Em C passa-se parâmetros simples para funções. Em C++, propõe-se a passagem de objetos como parâmetros dos métodos. Esta diferenciação proposta tem a sua razão de ser, já que poderia ter sido proposto um esquema de passagem de parâmetros para os métodos igual ao especificado no binding em C, tendo-se em vista que C++ é uma linguagem compatível com C. A principal razão que levou à adoção desta diferenciação foi a tentativa de tornar a interface API mais amigável para o usuário, uma vez que o uso da API é bastante complexo não só pelo número de funções que a API oferece, mas principalmente pelo número de parâmetros que estas necessitam como entrada.

Assim sendo, optou-se por uma solução que abstrai o usuário de parte destes parâmetros. Esta solução foi oriunda da observação de que, em geral, grande parte desses parâmetros assumia sempre valores default. Desta maneira, o usuário passa ao método um objeto. Este objeto contém os parâmetros de entrada e saída a serem utilizados por esse método. A grande vantagem reside no fato de que, desta maneira, podemos usar o mecanismo conhecido como constructor e disponível na maioria das linguagens orientadas a objetos, o qual permite a inicialização de um objeto no momento de sua declaração. Desta forma, todos os parâmetros de entrada recebem valores default, bastando ao usuário modificar aqueles relacionados com a sua necessidade específica.

Exemplificando, apresenta-se a seguir, em pseudo-C++, o caso em que o usuário ativa, executa operações e após desativa uma Application Entity (AE). Na realidade, a ativação de uma AE corresponde à ativação de dois processos, um correspondente à classe de objeto O_api e outro à O_mmpm, conforme explicado na seção VII.

```
:
:
:
Return_code              ret_code;
O_Library_calls          lib;
O_mm_aeactivation        ae (param1, param2);
```

```
O_mm_aedeactivation        deac;

main ( )
{
                .
                .
                .
        ret_code = lib.mm_aeactivation (ae);
                .
                .
                .
        deac = new O_mm_aedeactivation (param3,param4);
        ret_code = lib.mm_aedeactivation(deac);
                .
                .
                .
};
```

Como podemos observar no exemplo acima, a variável lib é do tipo O_Library_calls. Ae e deac são objetos usados como parâmetro nos métodos mm_aeactivation e mm_aedeactivation respectivamente. Os parâmetros param1, param2, param3 e param4 são valores não default que o usuário necessite modificar.

Os parâmetros de saída também estão incluídos nos objetos passados aos métodos. Desta forma, se o usuário quiser obter determinado status da operação mm_aeactivation, deverá fazer:

```
status = ae.inout_dcb->param;
```

O procedimento empregado sobre as demais funções da API, é análogo ao acima apresentado.

As modificações apresentadas permitiram-nos responder afirmativamente à questão da aplicabilidade da especificação de API à uma linguagem orientada a objetos. Estas modificações permitem ainda, ao nosso ver, uma simplificação no uso da API pelo usuário, pois este pode utilizar-se das facilidades incorporadas pelo uso dos constructors, conforme foi anteriormente explicado.

V.2. A Classe O_api

A classe O_api corresponde aos mecanismos citados em [GMM88]. Apesar de a API não seguir na norma utilizada os princípios da orientação a objetos, foi possível chegar-se à divisão em objetos representada na figura V. Nesta divisão O_api incorpora quatro objetos básicos: O_hlsp, O_csp, O_ifa e O_csp. Cada um destes corresponde a um dos blocos funcionais da parte inferior da API. Além destes quatro blocos funcionais, duas outras importantes classes de objetos integram a classe O_api: a classe O_apoio e a classe O_tabelas. A classe O_apoio provê métodos para a montagem, desmontagem e análise das PDUs MMS. Portanto, em nossa implementação, as PDUs são passadas já montadas para a MMPM, valendo o mesmo

no sentido contrário de comunicação. Já a classe O_tabelas engloba todas as tabelas utilizadas por O_api, fornecendo também os métodos para acesso às mesmas.

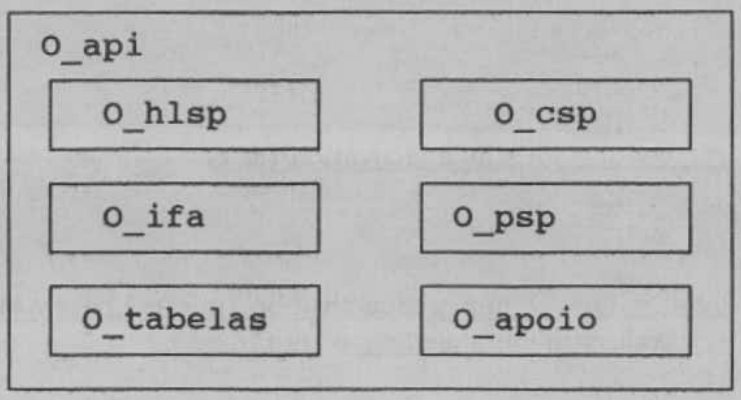


Figura V. Classe O_api

Entretanto, o esquema citado apresentava uma dificuldade. Em C++, um objeto herda características de um outro, de maneira que um objeto de um nível inferior não pode chamar métodos de objetos de níveis iguais ou superiores. É o problema da falta de bidirecionalidade, conforme explicado em [BUR92]. Portanto, adotou-se um esquema de maneira a permitir a troca de informações, em forma bidirecional, entre as classes de objetos O_hlsp, O_csp, O_ifa e O_psp, conforme fazem seus duais funcionais na especificação da API. Assim, foi também criada a classe O_endereços, a qual contém os endereços de O_hlsp, O_csp, O_ifa e O_psp sendo que cada um destas classes de objeto herda uma instância de O_endereços.

V.3. A Classe O_mmpm

A classe O_mmpm provê as funções da máquina de protocolos, MMPM (Message Manufacturing Protocol Machine) para a arquitetura Mini-MAP. Compõe-se, conforme observa-se na figura VI, de objetos do tipo O_tsm, que correspondem às TSM (Transaction State Machine). As funções desta classe de objeto foram modeladas em linguagem Estelle, de forma completa [BUR90a].

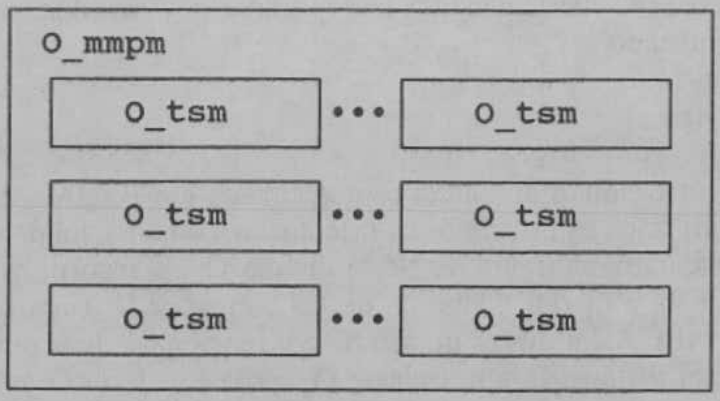


Figura VI. A Classe O_mmpm

A modelagem em linguagem Estelle revelou que boa parte dos estados desta máquina estava relacionada com gerência de conexões. Assim, na implementação em C++, optou-se por desconsiderar os estados relacionados com conexões, eliminando assim considerável overhead. Tal decisão acabou por se revelar válida, pautada na experiência do exemplo de aplicação implementada, descrita adiante. A implementação da MMPM encontra-se descrita em [NOV92].

VI. A Escolha da Linguagem Orientada a Objetos

Com a adoção do paradigma dos objetos, faltava, entretanto, decidir qual a linguagem a ser usada, tendo, como já mencionado, recaída a escolha sobre C++. Pra atingir esta decisão, três linguagens comerciais foram avaliadas: Smalltalk, Pascal Orientado a Objetos [BOR88] e C++ [BOR90a]. A escolha destas prendeu-se ao fato de serem estas, atualmente, as mais populares. Uma breve análise revelou que o uso de Smalltalk não era aconselhável, tendo-se em vista os seguintes fatores:

- A propalada ineficiência do código gerado por Smalltalk;
- Smalltalk ser mais voltada para sistemas comerciais e administrativos;
- Indisponibilidade de um sistema tempo real para o uso com Smalltalk.

Portanto, restavam Pascal Orientado a Objetos e C++. O estudo destas revelou que ambas se prestariam para uma implementação. Pascal incorporava a vantagem de ser bastante similar à Estelle. Desta forma, a tradução do código em Estelle para Pascal seria quase que imediato. Entretanto, optou-se por C++ baseado nas seguintes razões:

- Disponibilidade de um binding em C do API. Com isso, poderíamos tentar usar o mesmo, ou realizar pequenas modificações neste de forma a torná-lo mais compatível com a filosofia da orientação a objetos;
- Em automação industrial, C é largamente utilizado;
- C++ parece gerar um código bastante eficiente;
- Ambiente de depuração bastante completo [BOR90a];
- Disponibilidade de um sistema tempo real para C, compatível com C++ (MT-DOS [MIR90a]), conforme testes realizados.

Assim, optamos por C++. Entretanto, lembramos mais uma vez que Pascal também seria uma solução aceitável.

Sobre as características de C++, pode-se dizer que entre as linguagens orientadas a objetos, C++ vem se destacando em função da mesma incorporar toda a funcionalidade do C tradicional. Desta forma, C++ atua como uma extensão ao tradicional C, mantendo toda a eficiência deste. Portanto, programas anteriormente escritos em C podem ser aproveitados. Ao contrário de outras linguagens totalmente orientadas a objetos, como por exemplo Smalltalk, C++ tem caráter híbrido. Ao trabalhar-se com C++, pode-se adotar um enfoque de maneira a explorar-se totalmente o paradigma da orientação a objetos, ou um outro que encare C++ como um C que possui algumas construções adicionais. Por incorporar estas características híbridas, C++ vem tendo uma grande aceitação mundial. Entre as linguagens

Cs orientadas a objetos atualmente disponíveis, C++ parece o mais provável candidato a ser indicado como futuro padrão. Por outro lado, uma vez iniciada a implemantação de nosso protótipo, tivemos conhecimento do lançamento de Modula-3, linguagem que incorpora as facilidades de Modula-2 com a orientação a objetos. Fica aqui a sugestão de análise desta linguagem tendo em vista novos projetos, uma vez que com o uso da mesma não teríamos que usar um núcleo de sistema operacional a parte, simplificando a implementação.

VII. Mapeamento no Sistema Operacional

A escolha do sistema operacional adequado foi uma importante decisão de projeto. A especificação da API utilizada foi concebida pelo grupo MAP de forma a ser neutra com relação ao sistema operacional utilizado, tendo-se em vista que, à época, ainda não existia um sistema operacional padronizado pela OSI/ISO. Espera-se que o POSIX [FIP88] venha a preencher este espaço.

Segundo o citado documento, o único requisito que o sistema operacional deveria obedecer seria o fato de ser capaz de ativar e desativar processos. Entretanto, para facilitar a implementação das chamadas em modo assíncrono e síncrono, concluiu-se pela necessidade de um sistema operacional que oferecesse facilidades de definição e acionamento de semáforos, do tipo P e V. Em nosso protótipo, por questão de simplicidade não implementamos a facilidade, prevista no documento de espera sobre uma disjunção de eventos. Desta forma só é possível esperar por um evento a cada vez. A implementação da disjunção de eventos é passível de ser implementada no futuro, se interesse houver.

O sistema operacional selecionado foi o MT-DOS [MIR90a], por satisfazer bem as premissas acima e pela disponibilidade. Em nosso protótipo, as chamadas síncronas implicam que o processo requisitante fica suspenso em um semáforo, sendo este liberado finda a operação requisitada. Já no caso de chamadas assíncronas o usuário define qual o semáforo que ele gostaria de ser notificado ao término da operação. No momento em que o usuário deseja saber se a operação foi realizada ele efetua um "wait" sobre aquele semáforo, ficando suspenso caso a operação não tenha sido completada. Vejamos agora o mapeamento dos processos em classes de objetos, representada na figura VII, aonde as linhas pontilhadas representam processos e as cheias objetos.

O simulador de enlace utiliza o NT-DOS [MIR90b] para a troca de mensagens com outras estações, sendo estas trocas efetuadas sobre uma estrutura de comunicação Netbios/Ethernet.

Na citada figura, AP representa os processos de aplicação (Application Process). A classe de objeto O_Library_calls implementa as funcionalidades de Library Calls. O par O_api e O_mmpm fazem o papel das Application Entities (AEs).

As classes de objetos O_api e O_mmpm são instanciados, sendo mapeados em diversos processos do sistema operacional. Uma instância de objeto na realidade serve de código para um processo MT-DOS, em um mapeamento um a um (uma instância de objeto em um processo).

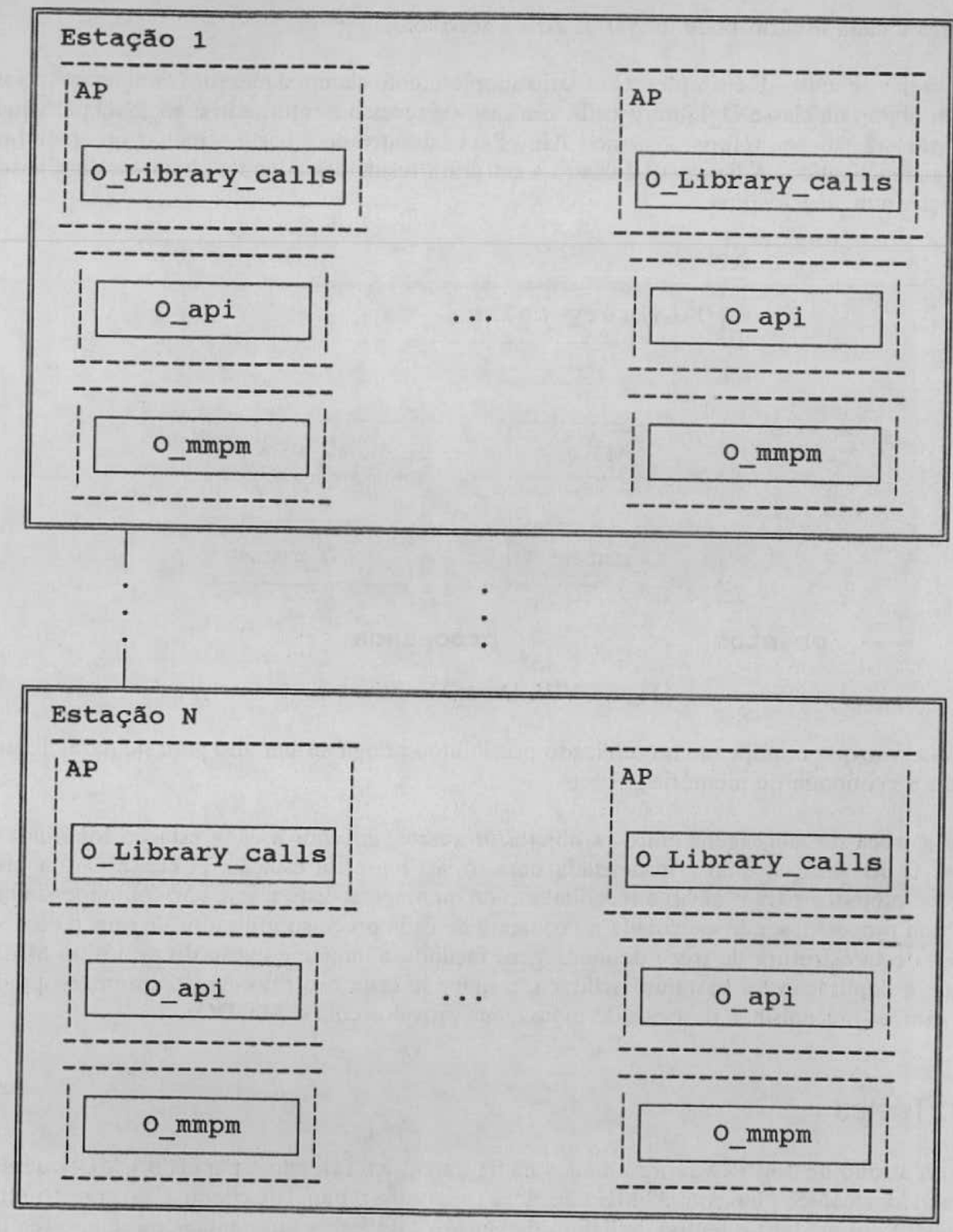


Figura VII. Mapeamento de Objetos em Processos

O mapeamento adotado, acarreta que cada AE (conjunto API com MMPM) só serve a um

usuário e cada usuário pode ter vários AEs a serví-lo.

A ativação de uma AE é simples. O usuário simplesmente chama o membro mm_aeactivation de um objeto da classe O_Library_calls. Em caso de sucesso é retornado o ae_label pelo qual ele passará a se referir o este AE. Para desativá-lo, basta chamar o membro mm_aedeactivation. A figura VIII ilustra a estrutura resultante após ser chamada duas vezes a função mm_aeactivation.

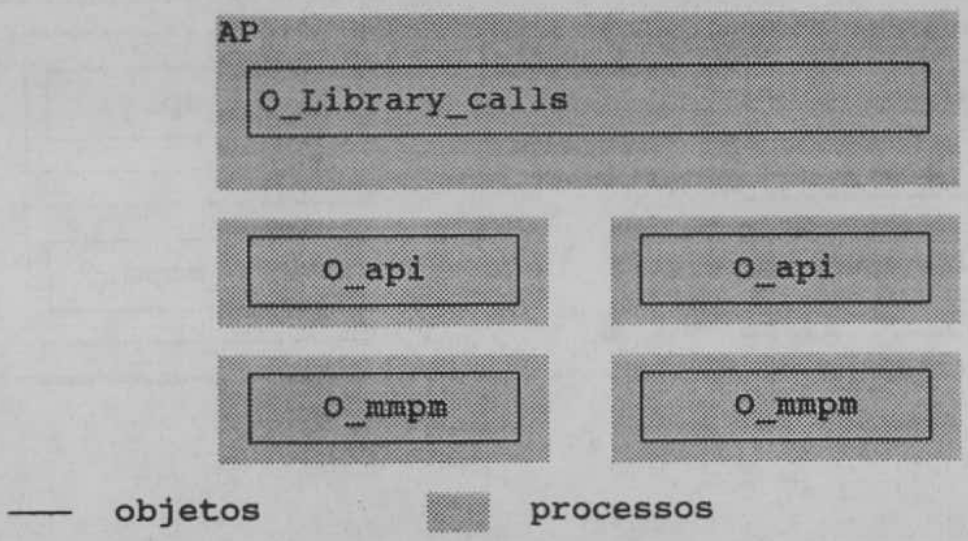


Figura VIII. Duas AEs ativadas

Pode-se ver que o mapeamento utilizado possibilitou atingir-se um alto grau de paralelismo aliado à economia de memória.

Para a troca de mensagens entre os objetos/processos internos a cada estação foi criada a classe O_fila_msg, a qual é instanciada uma só vez em cada estação. A classe O_fila_msg oferece métodos para o envio e recebimento de mensagens. Esta classe não foi mapeada em nenhum processo, sendo executada no contexto de cada processo utilizador de seus métodos. O uso desta estrutura de troca de mensagens facilitou a implementação do protótipo MMS, já que a depuração foi bastante facilitada, o que não teria ocorrido caso tivessemos optado por usar os mecanismos de troca de mensagens providos com o MT-DOS.

VIII. Testes

Com o auxílio de dois PCs representados na figura IX, um fazendo o papel de VMD, ou seja, um servidor MMS, (386 com 33 MHz de clock) e o outro o papel de cliente (286 com 16 Mhz de clock), foi medido o tempo de leitura de variáveis de forma automática, ou seja, o tempo em que uma chamada a primitiva de mm_read era completada. O tempo medido foi, em média, de 33 ms.

Este tempo é bastante apropriado para sistemas de tempo real [MEN89]. Em realidade nestes 33 ms estão incluídos 8 ms gastos pelo protocolo do sistema operacional de rede na troca de

mensagens, conforme tivemos oportunidade de medir. Assim se estivéssemos operando com um LLC real, certamente os tempos seriam menores. Por outro lado, acreditamos que os tempos poderiam ser bastante melhorados com algum esforço de otimização sobre o protótipo.

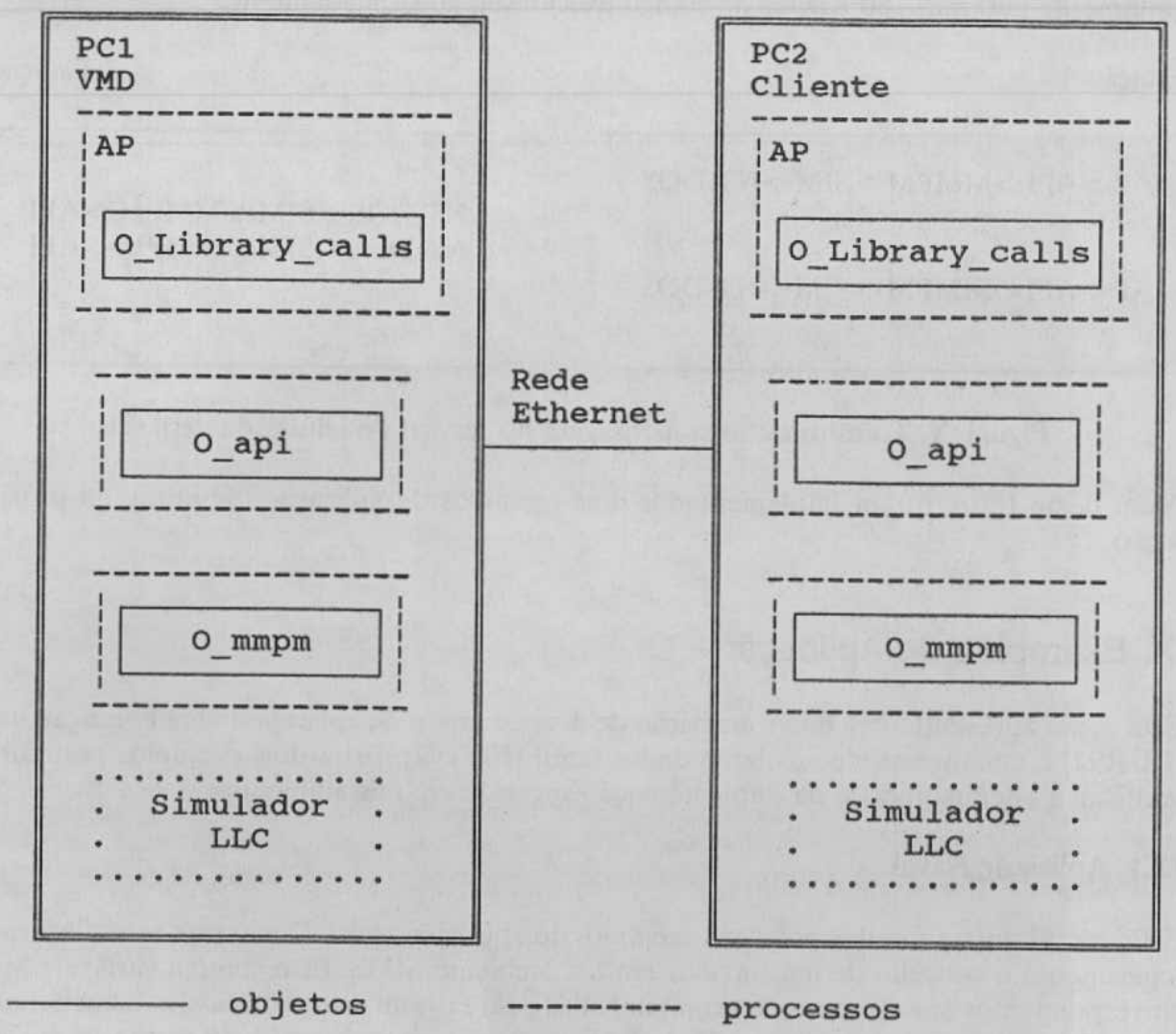


Figura IX. Ambiente de Testes

Caso se mostre necessária alguma otimização, podemos apontar como principal gargalo de nossa implementação o número de processos pelos quais uma mensagem tem de trafegar. Este número é oriundo do mapeamento dos objetos no sistema operacional utilizado, o qual obriga que uma mensagem para sair de uma estação passe por mais 3 processos além do AP gerador (cada processo na realidade recebe um ponteiro para a mensagem). Ou seja, na ida e na volta a mensagem acaba se deslocando por pelo menos 13 processos, 8 no lado do requisitante e 5 no lado do respondedor, conforme representado na figura X. Assim, em um ambiente que demande um desempenho superior, a sugestão de melhorarmos o procedimento de troca de mensagens entre os processos pode ser uma solução.

Em termos de linhas de código em C++, a implementação dos seviços mencionados em V.1,

no que se refere à API, foi efetuada em aproximadamente 4000 linhas, sem os comentários. A implementação da MMPM consumiu 1000 linhas aproximadamente (Em Estelle foram aproximadamente 2000, já que os estados relacionados com conexão foram também especificados). O conjunto API, MMPM e simulador de LLC compilados e linkados juntamente geraram 180 Kbytes de código executável, aproximadamente.

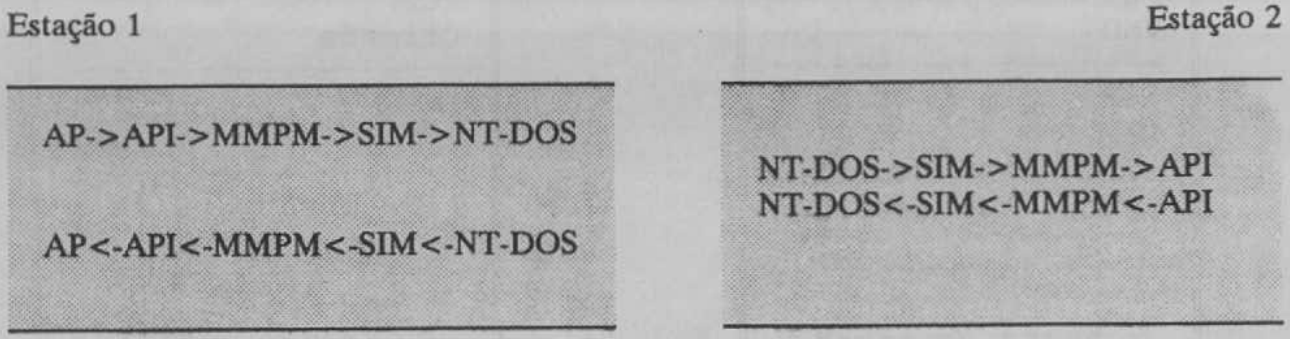


Figura X. Trânsito de uma mensagem no serviço de leitura de variáveis

Além deste teste, foram implementados dois exemplos de aplicação, descritos na próxima seção.

IX. Exemplos de Aplicação

Esta seção apresenta uma breve descrição de dois exemplos de aplicação: uma aplicação naval [BUR92] e um sistema de coleta de dados fabril [NOV92]. Estes dois exemplos permitiram verificar a adequabilidade da implementação sugerida em tais ambientes.

IX.1. Aplicação Naval

A figura XI ilustra um dos possíveis exemplos de aplicação naval. O extrator radar incorpora logicamente o conceito de um servidor em um ambiente MMS. Este contém variáveis MMS correspondentes aos alvos, sendo acompanhados pelo extrator radar. Cada alvo acompanhado é representado por um conjunto de variáveis, a saber: tipo, marcação e distância do alvo. Desta maneira, periodicamente, um processo da estação de acompanhamento tático acessa o servidor através do serviço ReadMMS, com a opção List, de forma a atualizar suas variáveis locais.

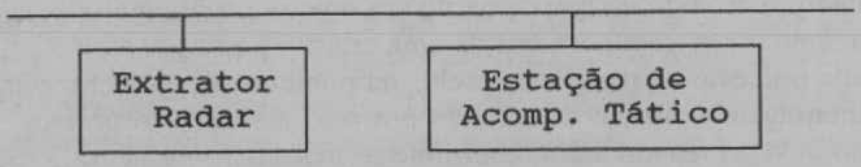


Figura XI. Exemplo de Aplicação Naval

Em se tratando de um extrator radar inteligente, este ao detectar o ataque de um míssil avisa a estação de acompanhamento tático através do serviço MMS de sinalização de eventos,

ReportInformation.

IX.2. Sistema de Coleta de Dados Fabril

O Sistema de Coleta de Dados tem como objetivo a geração de um banco de dados, a partir dos resultados dos testes realizados no produto, de forma que este posssa ser consultado posteriormente para a análise da situação da fábrica. Este sistema baseia-se numa rede local Mini-MAP, permitindo uma futura ligação com redes MAP-completo.

A utilização da norma MMS permite a transparência da execução dos vários tipos de testes nos produtos em relação à coleta dos dados, ou seja, a estação responsável pela realização dos testes funciona como um servidor MMS.

Na fase de protótipo definiu-se o subconjunto dos serviços MMS a serem implementados: invocação de programas, acesso a variáveis e gerenciamento de domínios.

X. Comentários

Neste trabalho foram apresentados o projeto e a implementação orientados a objetos de um protótipo do protocolo MMS. A estrutura resultante implementa três camadas do modelo de referência OSI, dentro da filosofia da chamada arquitetura Mini-MAP. O uso da mesma estrutura em uma implementação com as sete camadas do modelo OSI é facilmente realizável a partir dos conceitos aqui discutidos. Foram apresentadas também algumas modificações na especificação de API do grupo MAP, de forma a torná-la compatível com a filosofia da orientação a objetos.

Um exemplo de sistema naval usando o protocolo MMS foi descrito, dentro da proposta de uso do referido protocolo para tal classe de sistemas [BUR90b]. Apresentou-se também um sistema de coleta de dados fabril.

Referências Bibliográficas

[BOR88] Borland International. "Turbo Pascal - Object-Oriented Programming Guide",1988.

[BOR90a] Borland International. "Turbo C++, Getting Started & Programmer's Guide", 1990.

[BUR89] Burrowes F.B.. "Formalização do Processo de Avaliação de Redes Locais de Computadores", XVIII Jornadas Argentinas de Informática e Investigación Operativa, Buenos Aires, Agosto de 1989.

[BUR90a] Burrowes F.B., Novaes M., Azevedo M., Alves L., Pedroza A., "Modelagem, Implementação e Desenvolvimento de Aplicações em um Sistema Mini-MAP", V Congresso Latino-Ibero-Americano de Investigación Operativa e Ingenieria de Sistemas, Buenos Aires, 1990.

[BUR90b] Burrowes, F.B., Pedroza, A.C.P.. "Aspectos de um Sistema MMS para Aplicações Navais", Pesquisa Naval, Suplemento da Revista Marítima

Brasileira, 3ª edição, 1990.

[BUR92] Burrowes, F.B.. "O Protocolo MMS aplicado a Sistemas Navais", Tese de Mestrado, COPPE/UFRJ, Maio de 1992.

[CHI91] Chitman, E.. "Um Estudo sobre Métodos para o Desenvolvimento de Software", Tese de Mestrado, COPPE/UFRJ, Maio de 1991.

[COA91] Coad P., Yourdon E.."Object-Oriented Design", Yourdon Press, 1991.

[CRI91] Crispim, E.M.H.. "Avaliação de Métodos para o Desenvolvimento de Software", Tese de Mestrado, COPPE/UFRJ, Maio de 1991.

[DEA91] Dean, H.."Object-oriented design using message flow decomposition", Journal of Object-Oriented Programming, 1991.

[FIP88] Fips. "Posix: Portable Operating System Interface for Computer Environment U.S. National Department of Commerce/National Institute of Standards and Technology", Setembro de 1988.

[FON85] Fong, K. Amaranth, P.. "MAP Application Layer Interface and Application Layer Management Structure Computer Communication Review", Abril/Maio de 1985.

[GMM87] GM. "Map Specification", versão 3.0, Julho de 1987.

[GMM88] GM. "MAP-Appendix 7: Application Interface to MAP-Services", Agosto de 1988.

[IEE88b] IEEE 802.4. "IEEE Standard 802.4 - Token Passing Bus Access Method and Physical Layer Specification", Draft L, Agosto 1988.

[ISO87] ISO/DIS 9506."Manufacturing Message Specification, Part 1: Service Specification & Part 2: Protocol Specification", Draft 6, 1987.

[MEN89] Mendes, M.. "Comunicação Fabril e o Projeto MAP/TOP", IV EBAI, Escola Brasileiro-Argentina de Informática, 1989.

[MIR90a] Mira Informática. "Manual de Referência Técnica MT-DOS, Adaptador Multi-tarefa para DOS", Janeiro 1990.

[MIR90b] Mira Informática. "Manual de Referência Técnica NT-DOS", Janeiro 1990.

[NOV92] Novaes M.. "Projeto de um Sistema Industrial usando o Padrão MMS", Tese de Mestrado, COPPE/UFRJ, Maio de 1992.

[PED89] Pedroza, A., Valim, P., Goulart, C., Oliveira JR., R.. "Um Sistema de Auxílio ao Projeto de Protocolos de Comunicação para Redes de Computadores", Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos, Florianópolis, 1989.

[SHL88] Shlaer, S., Mellor, S.. "Object-Oriented Systems Analysis: Modeling the world in data", Yourdon Press, 1988.

[WIN90] Wing, J.M.. "A Specifier's Introduction to Formal Methods", IEEE Computer Setembro de 1990.