INSTITUTO SUPERIOR TÉCNICO

DESENVOLVIMENTO DE UM SISTEMA GENÉRICO DE GERAÇÃO DE CONSOLAS DE COMANDO E MONITORIZAÇÃO PARA ROBÓTICA OCEANOGRÁFICA

José João Nunes de Almeida nº 40163

Miguel Duarte Borges nº 40192

Responsável:

Professor Doutor António M. dos Santos Pascoal

Orientação:

Enge Paulo Jorge C. R. Oliveira

ISR - Instituto de Sistemas e Robótica

Lisboa, Setembro de 2000

ÍNDICE

SUMÁRIO

Este trabalho tem como objectivo o desenvolvimento de um sistema genérico de geração de consolas de comando e monitorização para robótica oceanográfica. Pretende-se um sistema que possibilite a visualização de dados, enviados por um veículo através de uma rede, com vista ao acompanhamento da missão. Para além disso deverá estar preparado para actuar sobre o referido veículo, criando tramas de comunicação e enviando-as.

A principal dificuldade do trabalho reside em conseguir uma abstracção em relação ao tipo de veículo a controlar, assim como em relação ao formato das tramas de comunicação. Cabe ao utilizador da aplicação a definição de configurações, consistindo estas especialmente em:

- Definição dos manómetros a usar e suas características, agrupados em separadores;
- Definição do formato das tramas de comunicação, nomeadamente dos campos constituintes;
- Associação entre os campos das tramas de comunicação e os manómetros definidos.

São estes três passos que definem o processo de geração de uma configuração de controlo. Caberá ao utilizador adequar essas configurações, consoante o uso específico que pretende dar à consola. A consola deverá ser o mais funcional e simples possível, garantir a visualização correcta e ordenada dos dados e fazer o logging das tramas recebidas para um ficheiro.

Durante o processo de desenvolvimento foram utilizadas diversas técnicas, nomeadamente ao nível do sistema de configuração das consolas a gerar e do mecanismo de descodificação/envio de tramas do/para o veículo.

Foram consideradas alternativas em relação ao modelo conceptual da aplicação, nomeadamente entre o uso de um ficheiro de texto onde o utilizador define a configuração pretendida de acordo com determinadas regras sintácticas e em relação a uma aplicação que usa interfaces visuais para realizar essa configuração. O segundo modelo acabou por ser adoptado devido às suas mais valias a nível de simplicidade de utilização.

Para representar os manómetros usaram-se controlos ActiveX, integrados na aplicação. As estruturas de dados foram definidas, tendo sido criadas diversas classes e estruturas para representar manómetros, tramas e associações entre elas.

A ferramenta de desenvolvimento usado foi o Microsoft Visual C++ 6.0 e o pacote gráfico escolhido para representar os manómetros foi o da Global Majic Software. O sistema de comunicação entre a consola e o veículo usa sockets do tipo UDP.

Todo o desenvolvimento foi efectuado no laboratório do grupo de Robótica Submarina do Instituto de Sistemas e Robótica (ISR), no Instituto Superior Técnico, no âmbito do trabalho final de curso dos autores.

AGRADECIMENTOS

Gostaríamos de agradecer ao Professor Doutor António Pascoal, pelo apoio concedido e por todas as condições de trabalho disponibilizadas, condições essas que nos proporcionaram um excelente ambiente de trabalho e aprendizagem.

Ao Engenheiro Paulo Jorge Oliveira, por toda a orientação prestada, pela sua constante disponibilidade e paciência em escutar as nossas dúvidas, pelas sugestões válidas que nos transmitiu e pelas críticas feitas que contribuíram para nos estimular cada vez mais.

Uma palavra de agradecimento também pela oportunidade dada de testar este trabalho nos Açores e pela experiência enriquecedora que constituiu o convívio com elementos do grupo de robótica submarina do ISR em Vila Praia da Vitória.

Ao Professor Doutor Carlos Silvestre pela simpatia e disponibilidade para tirar dúvidas, bem como pelos mesmos motivos, ao Engº Luís Sebastião, Engº Miguel Prado, Engº João Alves, Engº Carlos Ferreira e finalmente aos colegas Rui, João Trabuco e ao Telmo Azevedo.

Uma palavra muito especial de agradecimento e carinho aos familiares e namoradas por todo o apoio e paciência que nos proporcionaram ao longo do curso.

Para todos vocês o nosso obrigado,

Miguel Borges e José Almeida

1 INTRODUÇÃO

1.1 Nota Introdutória

Nas últimas décadas, verificou-se um grande desenvolvimento nas mais diversas áreas de conhecimento. A par deste desenvolvimento, no que diz respeito à robótica submarina, constata-se igualmente um aumento do número de expedições de pesquisa motivadas pela necessidade de recolha de dados necessários à investigação nos mais diversos domínios.

Os veículos desempenham nestas missões um papel fundamental. Estes podem deslocar-se onde o homem não pode ir pelos seus próprios meios, podendo por isso suportar condições de trabalho que seriam impossíveis à vida humana devido à adversidade inerente. Por outro lado, podem executar um conjunto de operações com uma precisão e rapidez muito superior àquelas que o homem está, do ponto de vista físico, apto a realizar.

Como exemplo de actividades onde estes veículos são particularmente úteis, podemos citar a Oceanografia, Geologia e Biologia Marítima, entre outros. O processo de concepção, desenho, engenharia e de fabrico deste tipo de veículos é naturalmente um processo moroso que exige um grande investimento do ponto de vista científico, humano e financeiro.

Durante o período de operacionalidade deste tipo de veículos, torna-se necessário poder controlar e/ou comandar de forma eficaz as operações a realizar, de modo a rentabilizar o investimento e garantir a segurança do próprio veículo.

1.2 Enquadramento do trabalho

Este trabalho foi motivado pelo projecto ASIMOV (Advanced System Integration for Managing the Coordinated Operation of Robotic Ocean Vehicles) do ISR – Instituto de Sistemas e Robótica – na área de Robótica Submarina, e tem como objectivo o desenvolvimento de um sistema genérico de geração de consolas de comando e monitorização para robótica oceanográfica, isto é, um sistema que nos permita actuar junto do veículo, para além de permitir a representação visual do seu estado. Esta interacção baseia-se nos dados recebidos através de uma rede ligada ao sistema que comunica com o veículo.

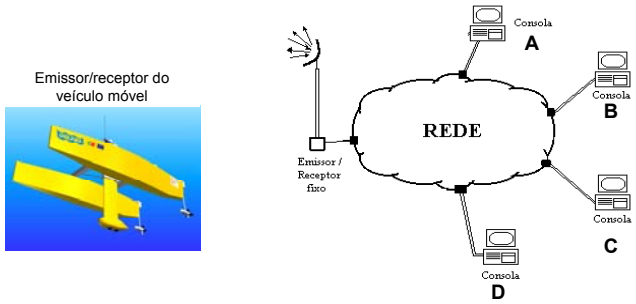

Fig. 1.1 – Veículo capacitado com sistema de transmissão de dados, emite para uma estação fixa que debita os dados numa rede a que as consolas acedem.

O exemplo da fig. 1.1 ilustra o esquema de um veículo que comunica por difusão com uma estação receptora/emissora que disponibiliza os pacotes de dados na rede onde diversas consolas estão ligadas.

A geração de uma configuração, realizada em *offline*, é feita fundamentalmente através de um processo de definição de tramas, de criação dos manómetros e sua configuração e pela associação dos campos das tramas aos manómetros desejados.

Ao receber tramas de dados num dado endereço IP:Porto, cujo formato foi previamente definido pelo utilizador, a consola descodifica essas tramas nos campos que as compõem. Opcionalmente podem-se aplicar transformações lineares (tipo $y = ax + b$) aos dados que compõem cada campo. O próximo passo consiste em refrescar os manómetros associados aos campos com o valor lido.

O processo de desenvolvimento de um sistema como o atrás descrito, envolve naturalmente a execução de um conjunto de actividades, tendo em vista não só a qualidade do sistema, como também a sua conclusão dentro do prazo estipulado, não esquecendo a necessidade de identificar de início alternativas para o modelo conceptual da aplicação a desenvolver.

Relativamente a este modelo, foi inicialmente especificado que seria composto por dois módulos principais: um com a finalidade de gerar um ficheiro de configurações, obedecendo a determinadas regras (script); outro com o objectivo de interpretar o script e gerar a interface gráfica propriamente dita.

Após o desenvolvimento de parte significativa do módulo gerador de scripts propusemos abandonar este modelo passando a incorporar tudo num único módulo, em que a configuração seria definida recorrendo a interfaces gráficas, solução essa aceite e usada no resto do projecto, enriquecendo dessa forma o processo de configuração. Abordaremos com maior profundidade este assunto em capítulo posterior.

1.3 Organização do relatório

No capítulo 2 descrevem-se os objectivos, os requisitos (funcionais, não-funcionais), a arquitectura do sistema e o suporte do trabalho onde se inclui a justificação das ferramentas de desenvolvimento adoptadas, assim como dos manómetros adquiridos. Para além disto, é descrita a metodologia seguida no desenvolvimento.

O capítulo 3 descreve as alternativas para o modelo conceptual da aplicação, nomeadamente um interpretador de script e uma configuração integrada, as técnicas utilizadas na implementação da interface gráfica, escolha da estrutura de dados e comunicações.

Os restantes capítulos dedicam-se à apresentação dos resultados obtidos, nomeadamente os testes realizados no banco D. João de Castro (Açores), terminando com uma apreciação ao trabalho desenvolvido.

2 CONCEITOS BÁSICOS

2.1 Objectivos

Pretende-se desenvolver um sistema que permita a geração de consolas de comando e monitorização de veículos. A aplicação deverá permitir ao utilizador definir a estrutura das tramas de dados provenientes da rede de sensores do veículo, a criação e configuração de manómetros e, por último, associar os campos das tramas definidas aos manómetros. Resumindo, pretende-se um sistema que permita a costumização de uma consola de acordo com as necessidades do utilizador. O sistema deverá ser desenvolvido para a plataforma Windows 95/98/NT™.

Deseja-se também que a aplicação seja:

1 - rápida
2 - robusta
3 - flexível
4 - simples
5 - funcional
6 - amigável

Deverá ainda ser capaz de gerar ficheiros de *logging* das missões monitorizadas, para além de garantir a sequência correcta na apresentação de resultados.

Apesar de não ter sido especificada qualquer orientação para o desenvolvimento da interface gráfica, considerámos importante dar alguma ênfase à interface homem-máquina, de modo que a interacção com o utilizador seja o mais amigável possível, sem que isso represente perda de operacionalidade ou funcionalidade.

2.2 Suporte do trabalho

Existem à partida duas componentes perfeitamente distintas:

a) o software que providenciará todo o ambiente de configuração para a interacção com o veículo e respectiva monitorização;

b) a rede através da qual o software comunica com o veículo.

Pretende-se que a aplicação através da qual se monitoriza e comanda o veículo, seja totalmente independente do seu processo interno de funcionamento. Deste modo, garante-se que uma eventual alteração nos algoritmos e arquitectura do veículo, não implica a redefinição da consola. A existir alguma alteração na estrutura das tramas enviadas/aceites pelo veículo, a consola deve estar preparada para reconfigurar a estrutura das tramas, isto é, de modificar a sua estrutura ou mesmo para adicionar novas tramas ou campos.

O computador de edição/compilação é uma plataforma munida do sistema operativo Windows 98.

Considerações gerais a ter em conta, referem-se a compatibilidades. A consola é executável em qualquer plataforma com um sistema operativo Windows 95/98/NT que permita o acesso a uma rede através de endereços IP:Porto. É aconselhado o Windows 98 ou Windows NT 4.0 por razões de compatibilidade gráfica, devido ao facto do software de edição/compilação ter sido desenvolvido em Visual C++ 6.0. A aplicação utiliza um pacote de manómetros que deverão estar instalados na plataforma de execução. Esse pacote consiste num conjunto de objectos OCX da Global Majic Software (GMS), ao qual nos referimos na secção 2.2.2 .

2.2.1 Ferramenta de Desenvolvimento

A linguagem de programação inicialmente proposta foi o C ou C++, para a qual nos foi sugerido o Borland C++ (versão 4.5) como ferramenta de desenvolvimento. No entanto, visto a aplicação exigir funcionalidades obtidas através da incorporação de tecnologia *ActiveX*, decidimos efectuar um estudo comparativo entre as duas principais ferramentas de desenvolvimento orientadas para objectos que suportam a tecnologia ActiveX: Borland Builder 4.0 e Visual C++ 6.0 da Microsoft. Tanto o Builder 4.0 como o Visual C++ 6.0, são linguagens de programação textuais

(e não linguagens visuais, apesar do que o nome pode sugerir), que utilizam um construtor GUI[1] para conceber interfaces gráficas melhoradas e de uma forma muito mais rápida.

Os requisitos/desenvolvimento desta aplicação mostraram ser necessário:

- Utilizar tecnologia de 32 bits baseada no Windows, de forma a aumentar o rendimento, acelerando o processo de programação de aplicações e o seu uso para os sistemas operativos Microsoft Windows98 e Windows NT Workstation.
- Suporte de tecnologia ActiveX
- Facilidade de reutilização de código e componentes de outras aplicações (Objectos OCX) para construir novas soluções.
- Recorrer a sofisticadas interfaces gráficas usando ferramentas muito poderosas, isto é, recorrendo a novos paradigmas visuais de elevada potência, criando assim com muito mais facilidade, uma interface gráfica mais amigável.

Por outro lado, estas duas ferramentas beneficiam de constante evolução, o que de certa forma privilegia os seus utilizadores, devido à facilidade de futuras actualizações (*upgrades*) e pela compatibilidade com aplicações desenvolvidas em versões anteriores. Beneficiam também de alguma versatilidade, no que respeita à plataforma de execução.

De seguida apresentamos um resumo do estudo efectuado para a escolha da ferramenta de desenvolvimento, no qual indicamos as principais características relevantes para a implementação do sistema pretendido.

[1] *Graphic User Interface*

Qualidades

Visual C++ 6.0	***Borland Builder 4.0***
• Produz aplicações rápidas e robustas de 32 bits para os sistemas operativos MS Windows95/98 e Windows NT • Apresenta-se com um compilador e *linker* incremental, com um método de reconstrução mínima, o que torna mais rápido o ciclo *editar-construir-detectar* erros em projectos de dimensão média /elevada. • Contém uma galeria de componentes "*Component Gallery*", que contribui para o aumento de produtividade através das capacidades de reutilização, servindo como localização central para objectos pré-construidos, incluindo classes C++ personalizadas, controlos OLE / OCX e os seus próprios componentes. • Oferece mais de 150 classes que poderão ser utilizadas como base para o projecto (*MFC - Microsoft Foundation Class*) • Os "AppWizards" permitem criar automaticamente aplicações personalizadas que vão ao encontro aos requisitos de algumas das fases do projecto. • A programação por objectos é simplificada pela existência de um visualizador ("*ClassView*"), o que permite ao utilizador a gestão dos projectos como um conjunto de classes relacionadas. • Excelente integração com as principais ferramentas de desenvolvimento da Microsoft, tais como: Microsoft Visual SourceSafe, Microsoft Visual Test. Estas serão nomeadamente úteis para o desenvolvimento em equipa e teste de controlos ActiveX. • Detecção de erros (debugger) *Just in Time* com janela de memória e informações sobrepostas, janela de variáveis, etc. • Grande suporte técnico e muita documentação • Verifica os últimos refinamentos ao standard C++ • Compilação superior em termos de eficiência e rapidez comparativamente ao Borland Builder 4.0 • Interface amigável	• Inclui um novo ambiente de desenvolvimento nativo, que permite criar aplicações para múltiplas plataformas: Windows 98,Windows NT, Windows 3.1, e DOS. • "Object Windows Library (OWL) 5.0" - esta é uma nova versão do Object Windows Library e com esta versão podem-se escrever aplicações funcionando verdadeiramente a 32 bit, podendo usar ambas as plataformas de 16 e 32 bits sem modificar o código fonte. • A OWL 5.0 também encapsula novas APIs do Windows, incluindo WinSock (TCP/IP), e MAPI (e-mail) entre outras • Inclui suporte para OCX (OLE Customer Controls) • Inclui um depurador gráfico integrado de 32 bit com editor de recursos integrados, com novos controles baseados em Windows 95 • Contém um editor alargado de diálogos, que proporciona suporte para os controles comuns baseados em Windows 95. • Verifica os últimos refinamentos ao standard C++ • Boa integração de ferramentas visuais e textuais • Personalização do IDE[2] • Interface amigável • Boa integração de ferramentas para o desenvolvimento de componentes-orientados • Inclusão de Install Wizards

Limitações:

Visual C++ 6.0	***Borland Builder 4.0***
• Aumento da dimensão do projecto torna a aplicação de desenvolvimento muito lenta. • "Developer Studio" não evoluiu de uma forma rápida como evoluiu o IDE, mas está muito bem estruturado	• Aumento da dimensão do projecto torna a aplicação de desenvolvimento muito lenta. • Suporte técnico muito fraco • Não inclui ferramentas para expansão a outras plataformas. • Não oferece grandes facilidades na criação e edição de controlos do tipo ActiveX

[2] *interface developer environment*

Conclusão:

Desta exposição podemos concluir que enquanto o Microsoft Visual C++ 6.0 tem fundamentalmente os seus pontos fortes na manipulação dos objectos e integração com outras ferramentas, o Borland Builder 4.0 tem a vantagem da sua aproximação ao RAD (Rapid Aplication Developer). O Borland apresenta muito mais facilidade na iniciação à sua utilização, mas peca pela falta de potência no que respeita à manipulação de objectos, funcionalidade esta que o Visual C++ 6.0 explorou.

Em termos de suporte e apoio técnico o Visual C++ 6.0 revelou-se superior face ao pacote da Borland. Em conversa com alguns programadores experientes que trabalharam com ambas as ferramentas, a opinião era claramente favorável ao Visual C++ 6.0, em termos de suporte técnico e integração com outras ferramentas. Por todas estas razões a nossa opção recaiu sobre o Visual C++ 6.0.

2.2.2 Manómetros

Dada a natureza da nossa aplicação, onde se pretende a representação de vários sensores ilustrativos do estado do veículo e a possibilidade de actuação sobre outros tantos parâmetros relativos ao seu comando, verificou-se de imediato a necessidade de utilização de vários manómetros que deveriam apresentar alguma flexibilidade na representação de dados, necessidade esta motivada pelo grande número de variáveis e de diferentes unidades de medida envolvidas. Relativamente aos controlos de comando houve igualmente a necessidade de diferentes representações de forma a uma fácil associação *forma – efeito*.

Procedemos então à identificação de diferentes tipos de manómetros/controlos e à tecnologia que tinham por base. Identificámos no mercado algumas soluções que nos pareceram bastante adequadas para o tipo de aplicação pretendida, nomeadamente manómetros e controlos cuja tecnologia subjacente era o ActiveX e o VBX 16 bits. Se por um lado os VBX 16 bits apresentavam algumas facilidades inerentes ao facto de trazerem o código fonte, possuíam limitações ao nível das funcionalidades oferecidas e ao nível da sua difícil articulação com a ferramenta de desenvolvimento entretanto escolhida: Visual C++ 6.0. A nossa preferência recaiu entre os seguintes pacotes de instrumentação e medida, baseados em tecnologia ActiveX:

IAL - Instrumentation ActiveX Library, da Global Majic Software

IP - Instrument Pack, da Iocomp Software

Para cada um dos manómetros e controlos destes pacotes foram realizados testes de rapidez, funcionalidade e eficiência em ambas as ferramentas de desenvolvimento. Falamos do Borland Builder 4.0 e o Visual C++ 6.0.

Este foi um processo moroso, dadas as dificuldades que encontrámos ao nível da integração de alguns destes manómetros nas aplicações (que criámos para testes) desenvolvidas nessas mesmas ferramentas.

Após a identificação das capacidades de cada um dos pacotes de manómetros e controlos, isto é, das funcionalidades e peso computacional que introduziam nas aplicações em run-time, verificámos que o pacote IAL da Global Majic Software se apresentava como a melhor escolha.

Por um lado os manómetros e controlos deste pacote (IAL) eram ligeiramente mais rápidos que os da IP. Para além de uma maior diversidade, o pacote IAL, permite a configuração de muito mais parâmetros de representação e interface que os da IP. Em seguida apresentamos alguns exemplos destes manómetros, de forma a que o leitor identifique claramente o tipo de controlos a que nos referimos.

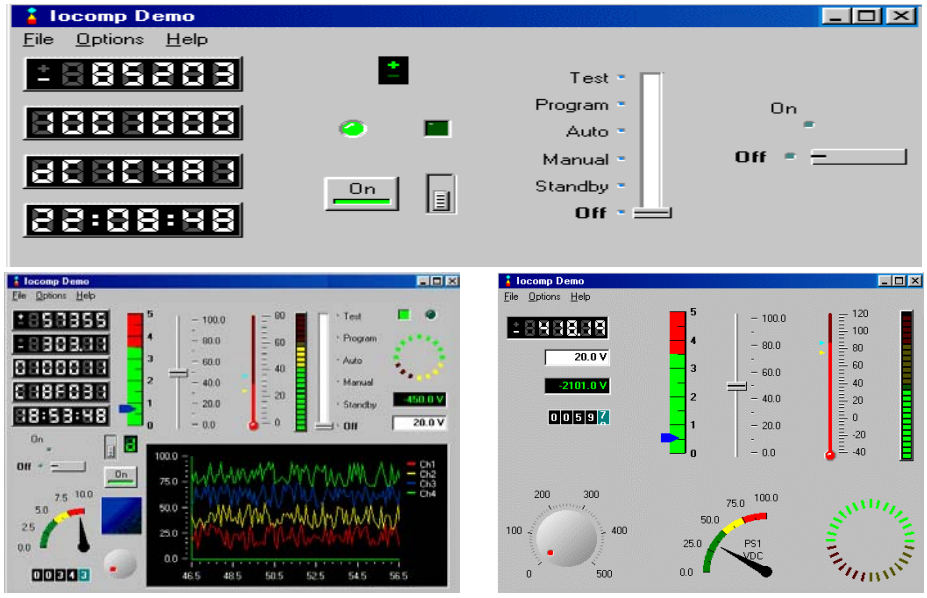

Fig. 2.1 Exemplos da IoComp

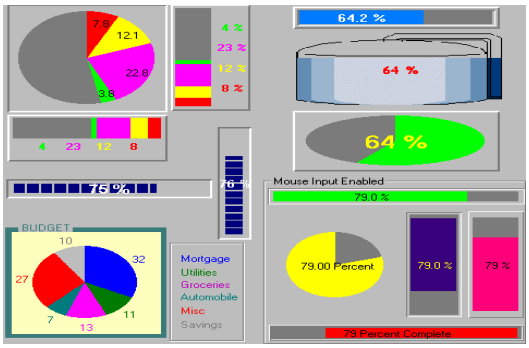

Fig. 2.2 – Strip Chart *(GMS)*

Fig. 2.3 – Instrumentos de navegação

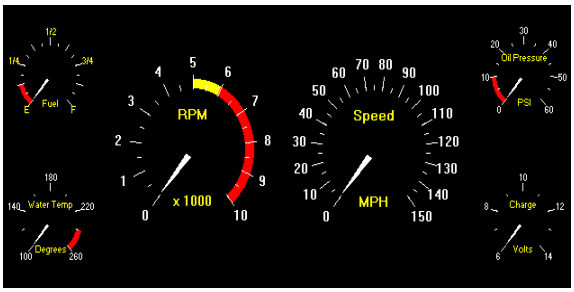

Fig 2.4 – Velocímetros, conta-rotações

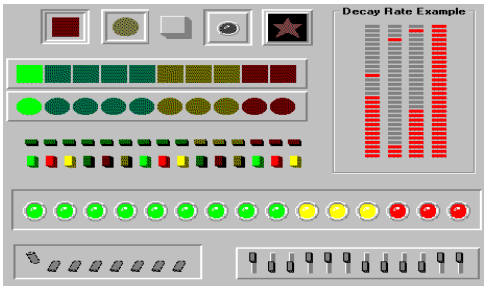

Fig 2.5 – LEDs *(GMS)*

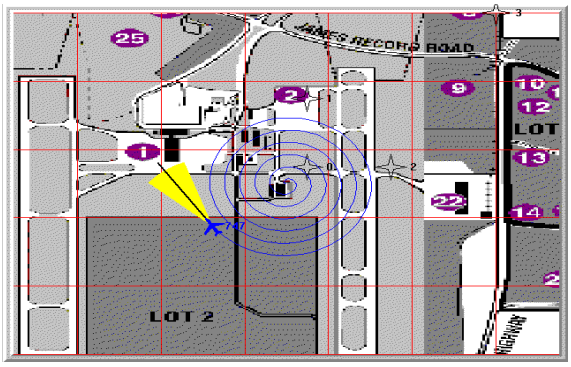

Fig. 2.6 – Moving Map *(GMS)*

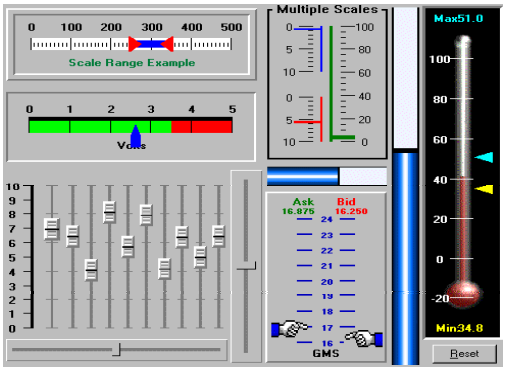

Fig 2.7 – Manómetros e selectores

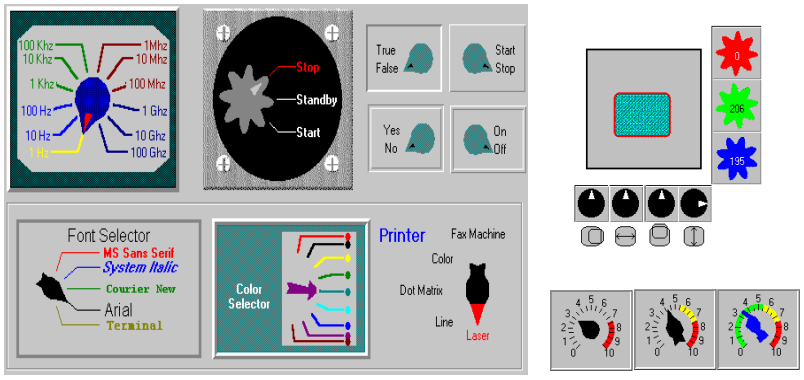

Fig. 2.8 – Selectores *(GMS)*

No fundo, estes OCX serão os elementos básicos da interface gráfica a desenvolver. Permitirão a representação e o comando através de interfaces adequadamente estudadas tanto do ponto de vista do comando como de representação de dados provenientes do veículo.

2.2.3 Suporte de comunicação com o veículo

Em virtude da especificação dada, o mecanismo de comunicação entre a consola e o veículo baseia-se nos sockets utilizando UDP[3]. O veículo com o qual a consola interage é virtualizado através de um endereço no formato IP:Porto.

2.3 Metodologia seguida

Em virtude do modelo de configuração inicialmente proposto, iniciámos um estudo sobre as ferramentas disponíveis em paralelo com o desenvolvimento de uma versão do gerador de script. Ainda antes de comprar qualquer pacote de manómetros, experimentámos uma integração com a script, usando uma demo da empresa GMS, a qual funcionou perfeitamente. Uma vez confirmada a qualidade do produto da GMS, seguiram-se contactos com vista à sua aquisição.

Foi neste ponto que propusemos uma alteração ao modelo inicialmente proposto, alteração essa que, após discussão, foi aceite por ambas as partes devido às mais valias que daí resultavam. Seguiu-se um período de reflexão com o objectivo de definir as estruturas de dados a usar. Paralelamente, dedicámos algum tempo à ambientação com a ferramenta de desenvolvimento escolhida. A partir deste momento definimos algumas etapas a cumprir, nomeadamente:

1. Definição de um protótipo da interface
2. Implementação das estruturas de dados
3. Desenvolvimento do núcleo funcional da aplicação com vista à criação e manipulação das estruturas de dados (criação de manómetros, tramas e associações)
4. Investigação do método adequado para proceder à comunicação e refrescamento dos manómetros (usando *threads* e *sockets*)
5. Implementação da estrutura de comunicação

[3] User Datagram Protocol

6. Desenvolvimento de uma aplicação auxiliar com vista à criação de um gerador de tramas destinadas à consola
7. Estabelecer a comunicação entre os sistemas e efectuar o refrescamento dos dados para um manómetro
8. Generalização para vários manómetros
9. Sincronização e *logging*
10. Teste em laboratório
11. Teste principal no banco D. João de Castro

O nosso principal objectivo foi ter uma versão funcional pronta a ser testada na saída, do grupo de robótica submarina, ao banco D. João de Castro (etapa 11) pelo que todos os esforços foram dirigidos para que tal fosse uma realidade.

3. TÉCNICAS UTILIZADAS

3.1 Identificação de Alternativas

Tal como referido anteriormente, o modelo de configuração inicialmente proposto estabelecia que a configuração da consola se processasse através da criação de um script a partir de um ficheiro gerado pelo utilizador, onde se definia o formato das tramas, a criação de manómetros e a sua associação aos campos das tramas definidas.

Entretanto decidiu-se, com a aprovação do Professor Orientador, enriquecer o sistema de configuração, evoluindo para um sistema de configuração integrado na própria aplicação. Esta decisão implicou uma avaliação prévia do impacto que a alteração ao sistema de configuração teria no processo de desenvolvimento, nomeadamente em relação aos prazos inicialmente estabelecidos, como também no que diz respeito à eficiência do processo de configuração.

Procedeu-se então à redefinição do modelo conceptual da aplicação, e consequentemente à redefinição parcial do processo de desenvolvimento, já que o formato das tramas, os manómetros e a sua associação aos campos das tramas será, neste caso, definido e alterado directamente na aplicação através de um sistema de diálogos que será discutido mais à frente.

Antes de prosseguirmos para a descrição do modelo conceptual associado às alternativas identificadas, indicam-se em seguida os principais passos do desenvolvimento da aplicação:

1. Estudo e escolha das ferramentas de desenvolvimento disponíveis.
2. Processo de selecção dos manómetros existentes no mercado com vista à sua incorporação na aplicação.
3. Criação do sistema de configuração:
   - 3.1. 1ª alternativa: interpretador de script
   - 3.2. 2ª alternativa: configuração integrada (Visual)
4. Desenvolvimento do serviço de gestão de tramas e tratamento de informação:
   - 4.3. Definição de estruturas de dados necessárias.
   - 4.4. Definição do modo de comunicação com o servidor de tramas.
   - 4.5. Definição do processo de descodificação e gestão da informação proveniente das tramas

5. Implementação e integração com a interface gráfica.
6. Teste.
7. Manutenção

NOTA: Lembramos que, em qualquer uma das alternativas propostas, o utilizador está restringido aos manómetros existentes que foram apresentados anteriormente.

3.1.1 Modelo Conceptual para a 1ª alternativa - Script

Segundo este modelo, a aplicação será construída tendo por base os seguintes módulos:

1. Interpretador de Script
2. Criação da interface Homem-Máquina
3. Processo gestor de tramas recebidas
   3.1. Comunicação com o Servidor de tramas

Apresentamos em seguida um esquema básico destes módulos:

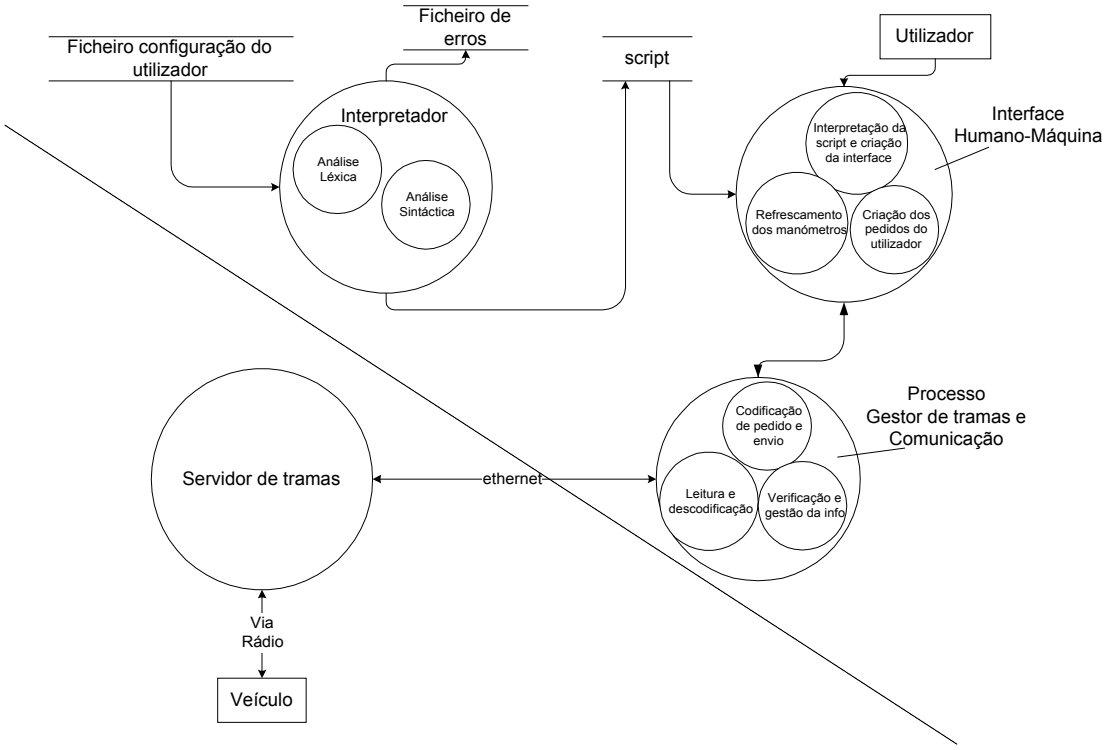

Fig. 3.1 –Elementos do modelo do interpretador de script e o seu esquema de ligação

Assim, os principais passos associados ao desenvolvimento desta alternativa são:

1. Criação de um interpretador para gerar um script de configuração válida:
   1.1. Criação do interpretador de forma independente de modo a testar a detecção de erros léxicos e sintácticos dos ficheiros criados pelo utilizador (neste passo utilizaram-se as ferramentas LEX e YACC como analisadores lexical e sintáctico respectivamente).
2. Desenvolvimento da aplicação, consistindo nas seguintes fases:
   2.2. Criação da interface gráfica através da interpretação do script gerado anteriormente
   2.3. Criar um manómetro a partir do script interpretado, bem como definir as tramas (também interpretadas a partir do script) e associar os seus campos ao manómetro
   2.4. Testar correcto funcionamento
   2.5. Generalização para geração de vários manómetros.

Na próxima secção descrevem-se os passos principais executados na implementação daquela que constituiu de início a única alternativa: o interpretador.

3.1.1.1 Descrição do Interpretador de Script implementado

O utilizador terá de criar um ficheiro de texto com a configuração pretendida, nomeadamente em relação à constituição das tramas da aplicação e do número e tipo de manómetros, para além da relação de associação entre as tramas e os manómetros.

Para gerar um ficheiro correcto e livre de erros foi desenvolvido um pequeno interpretador cujo objectivo é verificar a correcção do ficheiro do utilizador e gerar um ficheiro com o formato pretendido. A finalidade deste interpretador não é mais do que fazer a análise léxica e sintáctica do ficheiro fonte, desenvolvido pelo utilizador, detectando eventuais erros e gerando um ficheiro binário com as estruturas representativas das tramas e dos manómetros pretendidos.

Este ficheiro será posteriormente lido pela aplicação e criados os objectos representativos dos manómetros e das tramas.

O objectivo é permitir ao utilizador abrir um ficheiro com o formato correcto, isto é, tratado pelo interpretador ou, caso não esteja no formato pretendido, invocar o interpretador para gerar o referido ficheiro.

Nesta fase inicial o interpretador é independente da aplicação, pelo que o próximo passo consistiu na sua integração. Para a construção do interpretador foi usada uma ferramenta freeware disponível na internet em [18] designada "Parser Generator", que não é mais do que o Lex e Yacc, versão IDE para o Windows™. Esta ferramenta é desenvolvida para gerar código C/C++ a ser integrado em aplicações do Borland C++ e Visual C++.

3.1.1.2 Sintaxe

A sintaxe escolhida para o interpretador consiste numa zona de declarações das tramas e numa zona de declarações dos manómetros.
De seguida apresenta-se a gramática do interpretador em notação BNF (*Backus Naur Form*):

```
script ::=  declaração_de_tramas declaração_de_manómetros
```

A zona de declaração de trama começa com a palavra reservada "FRAME" e é delimitada pelos símbolos '[' e ']'.

```
declaração_de_tramas ::= FRAME [ lista_tramas ]
```

A definição de várias tramas é separada por ';' . A última trama não tem ';'

```
lista_tramas ::= trama |
                 lista_tramas ; trama
```

Uma trama é constituída por um identificador, seguido por um inteiro representativo da taxa de refrescamento, e os seus campos:

```
trama ::=  identificador , numero_int , campos_trama
campos_trama ::=   campo_trama  |
                   campos_trama , campo_trama
```

O campo da trama é constituído pelo seu identificador, um inteiro representativo do seu comprimento em bytes, e o seu tipo

```
campo_trama ::=  identificador  : numero_int : tipo
tipo ::=  CHAR | INT | FLOAT | BYTE
```

A zona de declaração de manómetros começa com a palavra reservada "GAUGE" é delimitada por '[' e ']'

```
declaração_de_manómetros ::= GAUGE  [ lista_manometros ]

lista_manometros ::=   manometro |
                       lista_manometros ; manometro
```

A declaração do manómetro é constituída pelo seu identificador, o identificador da trama associada ao manómetro e o seu tipo

```
manometro ::=  identificador  ( identificador_trama ) tipo_manometro
```

O tipo do manómetro é constituído por uma palavra chave correspondente seguido pelos campos desse tipo. Incluímos em seguida a notação apenas para um manómetro T1:

```
tipo_manometro ::=  _T_1 variaveis_man1

variaveis_man1 ::=   variavel_man1 |
                     variaveis_man1 variavel_man1
variavel_man1 ::=    X1          : numero_int    |
                     _X2         : numero_int    |
                     _Y1         : numero_int    |
                     _Y2         : numero_int    |
                     T_IV        : numero_real   |
                     T_DELTA     : numero_real   |
                     T_MIN       : numero_real   |
                     T_MAX       : numero_real
```

O identificador da trama é constituído por um identificador da trama e um identificador do campo da trama, separados por um ':'

```
identificador_trama ::= identificador  : identificador
identificador ::= caracter alfanum

alfanum ::= numero_int |
            caracter
numero_int ::=  digito |
                digito numero_int
numero_real ::= numero_int |
                numero_int . numero_int
digito ::= 0|1|2|3|4|5|6|7|8|9
caracter::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|y|x|z |A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|Y|X|Z
```

3.1.1.2 Exemplos

Ilustramos aqui através de um exemplo o seu modo de funcionamento

É usado o Notepad™ para editar um ficheiro de texto *exemplo1.txt*. Este ficheiro é passado ao interpretador que o converte no formato correcto lido pela aplicação gerando, deste modo, o manómetro.

Vamos criar um ficheiro com uma trama e um manómetro associado a essa trama:

A trama definida é a trama Watson, com uma taxa de refrescamento de 2 ms e com 3 campos inteiros.

-Manómetro Xpto do tipo 1 que representa os dados do 2º campo da trama Watson.

- Posição: left = 0 ; right = 100; top = 0; bottom = 450
- Valor inicial: 5.5
- Gama de Medida : min = -15; max = 50; intervalo de variação = 10;
- Inicio da escala (amarela) = 5

Lembramos que o interpretador de script identifica eventuais erros sintácticos, para além de verificar a validade de cada um dos campos de cada manómetro (erros semânticos).

Ficheiro de texto criado pelo utilizador com as especificações (sem erros):

```
FRAME
[
Watson , 2, campo1:2:int , campo2:2:int, campo3:2:int

]

GAUGE
[
XPTO (Watson:campo2) _T1 _left:0  _right:100 _delta:10 _bottom:450 _top:0 _initv:5.5  _min:-15 _max:50 _yellow:5
]
```

Fig. 3.2 – ficheiro texto criado pelo do utilizador com vista a definir uma trama e um manómetro

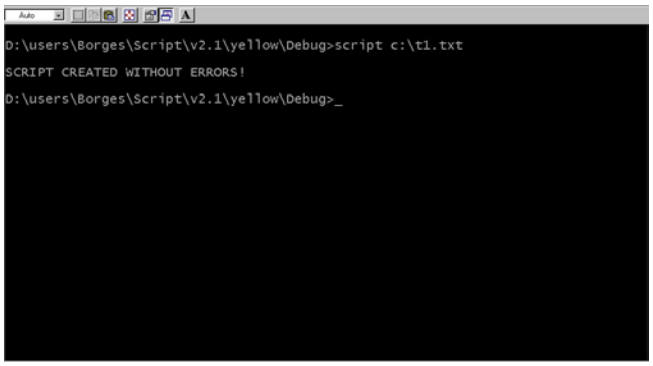

Fig. 3.3 – Output gerado pelo interpretador

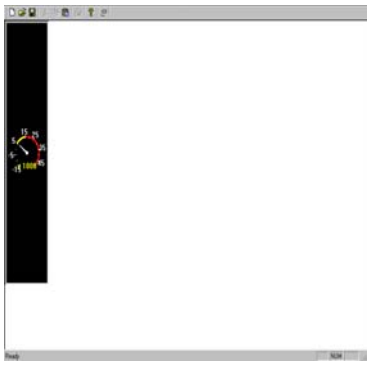

Fig. 3.4 – Manómetro

Alteração da posição do manómetro:

```
FRAME
[
Watson , 2, campo1:2:int , campo2:2:int, campo3:2:int

]

GAUGE
[
XPTO (Watson:campo2) _T1 _left:400  _right:700 _delta:10 _bottom:450 _top:250 _initv:5.5  _min:-15 _max:50 _yellow:0
]
```

Fig. 3.5 – Alteração do ficheiro de texto inicial

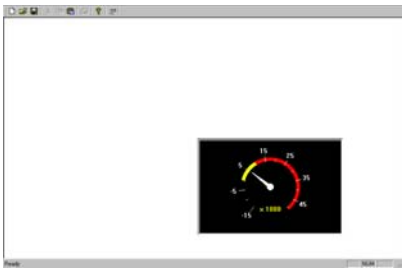

Fig. 3.6 – Posição alterada

O interpretador permite a detecção de erros sintácticos e semânticos. A figuras 3.7 e 3.8 ilustram a detecção deste tipo de erros

```
FRAME
[
Watson , 2, campo1:2:int , campo2:2:int, campo3:2:int

]

GAUGE
[
XPTO (Watson:campo2) _T1 _left:0  _right:100 _delta:1 _bottom:450 _initv:200  _min:-15 _max:50 _yellow:5
]
```

Fig. 3.7 - utilizador não preenche o um campo de um manómetro tipo 1

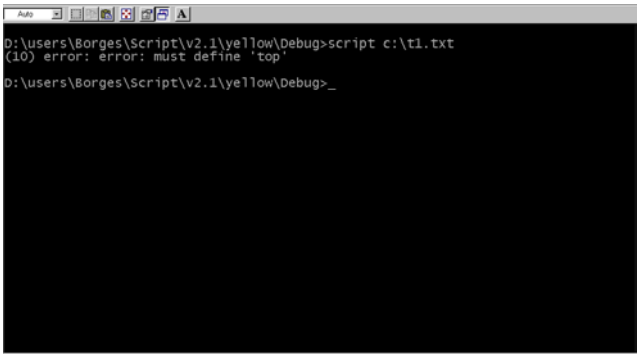

Fig. 3.8 – Resultado do interpretador

Todo este processo é moroso e complicado, dificultando a criação/edição dos ficheiros. Passamos em seguida à consideração da segunda alternativa.

3.1.2 Modelo Conceptual para a 2ª alternativa - Configuração Integrada

Neste modelo, a aplicação será construída tendo por base os seguintes módulos:

1. Interface Homem-Máquina
   1.1. Inclusão de um mecanismo de configuração baseado na API[4]
2. Processo gestor de tramas recebidas
   2.2. Comunicação com o Servidor de tramas através de uma rede de dados.

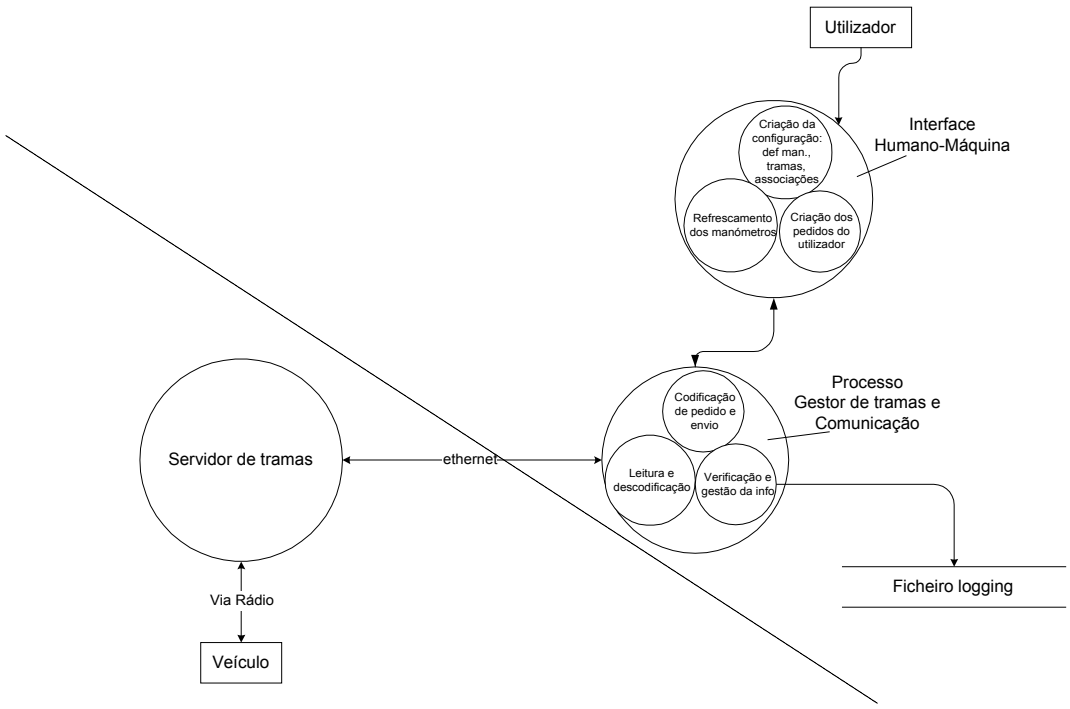

Fig. 3.9 - Elementos do modelo de configuração integrada

Os principais passos associados ao desenvolvimento de um sistema de configuração integrada são então:

1. Criação da operação de definição de tramas utilizando a API do Windows
2. Criação da operação de criação de manómetros e respectiva configuração utilizando a API do Windows.
3. Criação da operação de associação dos campos da trama aos manómetros já criados utilizando a API do Windows.

[4] *Application Programming Interface*

4. Implementação e integração de todas estas operações como funcionalidades da consola

3.1.3 Comparação e escolha das alternativas identificadas

Se compararmos ambos os modelos, podemos verificar que o sistema de configuração integrado está embebido na própria aplicação, excluindo desta forma o interpretador de script tal como apresentado anteriormente.

As grandes vantagens do modelo de configuração integrada, face ao interpretador de script, são a sua elevada flexibilidade, robustez, facilidade de criação e alteração de configurações, rapidez de realização, a sua integração na própria aplicação, a rapidez e por último a sua interface com o utilizador constituir uma abordagem mais amigável.

No nosso entender, apesar de entretanto já ter sido dispendido um esforço considerável no desenvolvimento do interpretador de script, esta alternativa apresenta-nos apenas vantagens relativamente à primeira, constituindo por isso uma mais valia para o sistema a desenvolver. Embora a mudança de estratégia implicasse uma sobrecarga do trabalho a desenvolver e um reajustamento dos prazos estipulados para o cumprimento de alguns objectivos, esta foi aceite como solução definitiva para o modelo de configuração a utilizar.

Resumo*:*

Nesta secção, apresentámos as alternativas quanto ao modelo de configuração, descrevendo o trabalho efectuado e as soluções encontradas relativamente ao modo de configuração inicialmente especificado (o interpretador de script); por outro lado introduzimos ao leitor o modo de configuração integrado que passaremos a descrever de seguida.

3.2 Interface Gráfica da Consola

O ponto central deste capítulo prende-se com o estudo e desenho da interface, e com as técnicas de interacção com a consola. Assim, na base de algumas opções tomadas relativamente à interface, estiveram alguns critérios que são identificados nas próximas páginas. Serão apresentados exemplos elucidativos da aplicação desses mesmos critérios na implementação da interface.

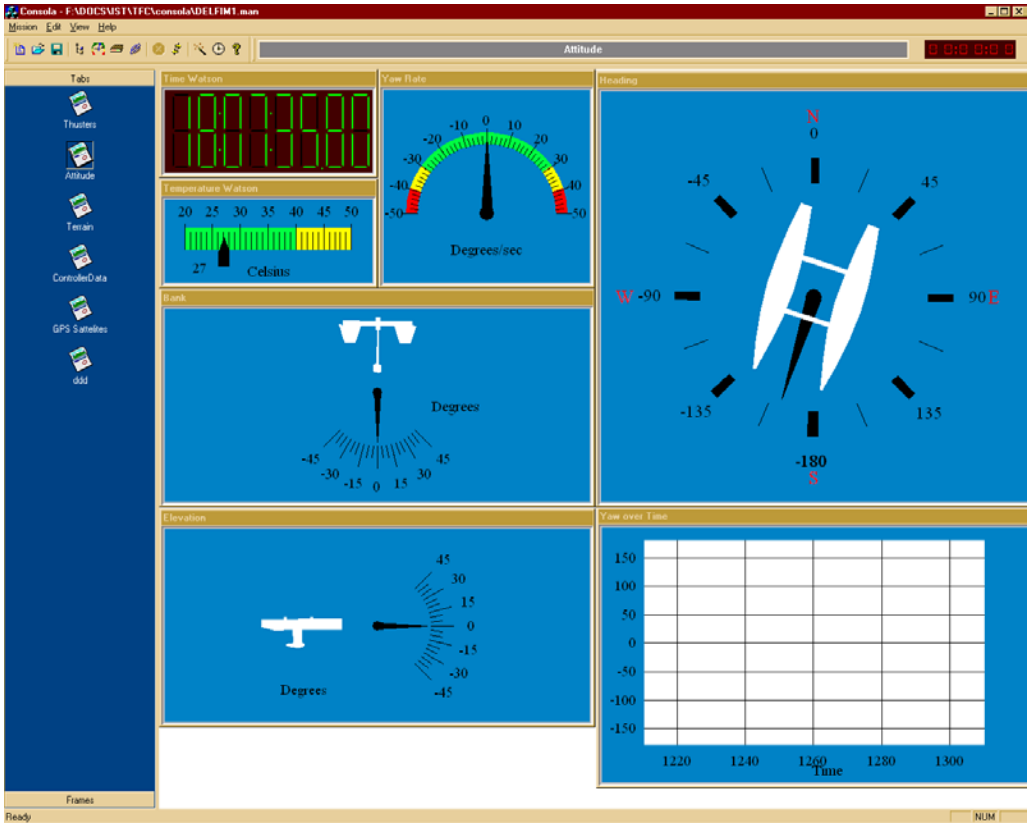

Fig. 3.10 – Exemplo de uma consola gerada

Sendo assim, os principais passos seguidos foram: 1 – Estudar a interface adequada para a troca de informação entre a aplicação e o utilizador; 2 – Análise das tarefas principais a implementar: criar manómetros, tramas e associações; 3 – Iniciar e terminar monitorização; 4 – Desenvolver e refinar protótipo de interface até à versão final; 5 – Integrar interface e núcleo funcional numa só aplicação.

3.2.1 – Descrição dos elementos básicos da interface

3.2.1.1 Separadores

Os manómetros quando criados através do processo de configuração, são inseridos num ambiente que foi cuidadosamente estruturado de modo a oferecer ao utilizador um ambiente amigável e bastante flexível, procurando ao mesmo tempo simplificar o processo. Decidimos então utilizar um estilo de diálogo simples e eficiente para cada um dos passos de configuração e para aplicação em geral, tentado dessa forma tornar a navegação mais eficaz.

Para tal, foi implementado um mecanismo de 'separadores' que permitisse agrupar, de acordo com as necessidades do utilizador, os manómetros em localizações diferentes. No fundo, cada separador representa uma vista, e em cada instante só um vista pode estar activa, isto é, visível ao utilizador.

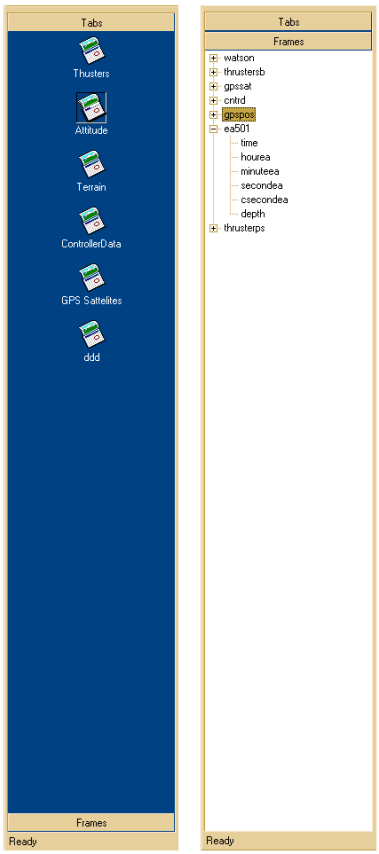

Fig.3.11 - Separadores e Estrutura de Tramas

Estes separadores são agrupados verticalmente na janela que se encontra do lado esquerdo da aplicação (Fig. 3.11). Cada separador é representado por uma pasta e por um nome que o identifica. Pode ser criado em qualquer altura sem obrigar a qualquer sequência de acções. Não existe limite para o número de separadores. Para consultar as tramas criadas, basta ao utilizador selecionar o grupo *Frames* na barra.

As funcionalidades associadas a cada separador são a mudança do nome, remoção e o seu reposicionamento na barra onde se encontram. A interacção com esta barra de separadores é feita através do rato para a selecção da operação a realizar, e do teclado para alteração do nome do separador. A dimensão dos seus limites é ainda alterável, bem como a dimensão dos ícones representativos dos separadores existentes. Podemos observar a Fig 3.12, onde, clicando com o botão direito do rato em cima da zona de separadores, surge um menu sugestivo, que permite ao utilizador remover separador, alterar nome ou alterar o modo de visualização dos ícones. Para alterar a ordem dos

separadores, o utilizador apenas tem de recorrer ao *drag&drop*, arrastando separador pretendido para a posição desejada.

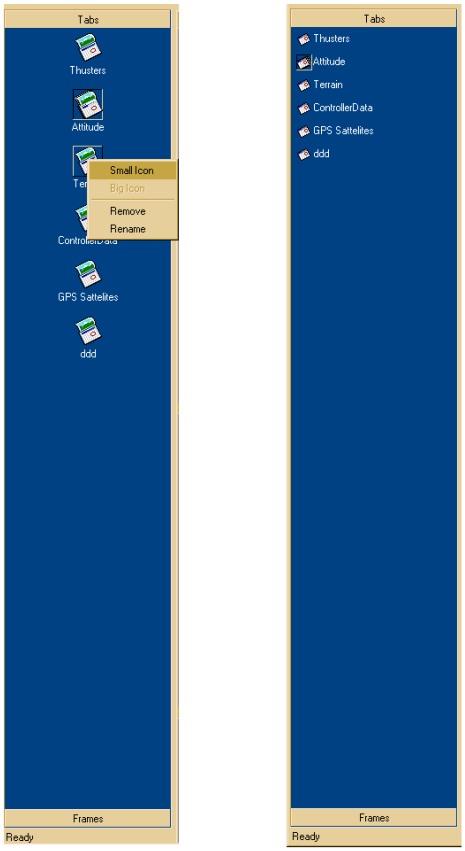

Fig. 3.12– Opções sobre separadores

Uma alternativa a esta barra seria incluir um menu de selecção de separadores. Apesar de ter a vantagem de não ocupar tanta área de visualização que poderia ser usada por manómetros, revela-se pouco prática do ponto de vista da usabilidade que pretendemos explorar ao máximo. Esta barra é ainda utilizada para descrever a estrutura das tramas conhecidas pela consola em cada instante.

Tal como explicado anteriormente, limitamo-nos a implementar as funcionalidades básicas de forma a obter uma interacção simples e eficiente. A localização dessa barra foi estudada tendo em vista não só a organização do interface, mas também na perspectiva de explorar a habituação dos futuros utilizadores deste sistema em usar este modelo de selecção para vistas diferentes. Essa organização foi, tal como será explicado mais à frente, baseada no agrupamento lógico das operações da consola. Assim, decidimos colocá-la do lado esquerdo da aplicação

3.2.1.2 Barra de comandos

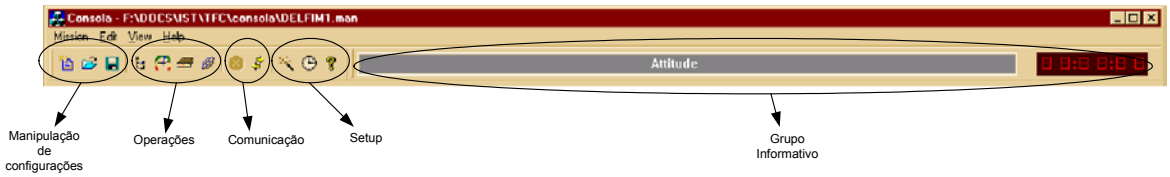

Fig. 3.13 – Barra de comandos da consola

Um outro elemento participante da interface da aplicação é a barra de comandos (Fig. 3.13). Esta agrupa um conjunto de comandos e de fontes de informação estando dividida em cinco grupos lógicos do ponto de vista das operações por cada um deles efectuadas, sendo esse agrupamento

acompanhado da mesma tonalidade de cor com vista a não distrair o utilizador e não tornar os próprios ícones confusos:

- Manipulação de configurações
- Operações
- Comunicação
- Setup
- Grupo informativo

As opções mais importantes, disponíveis na barra de comandos, estão também acessíveis ao utilizador através dos menus da aplicação. No entanto o objectivo desta barra é estimular o utilizador a não perder tempo a aceder aos menus, dispondo para isso de uma interface mais eficaz. Houve um cuidado especial na escolha dos ícones e das palavras do menu *pull-down* associando-lhes um vocabulário familiar. Pretendeu-se também uma comunicação clara e não ambígua tentando ir o mais possível, ao encontro do *Look & Feel*, ou seja, reflectir um dado comportamento numa dada aparência

Manipulação de configurações

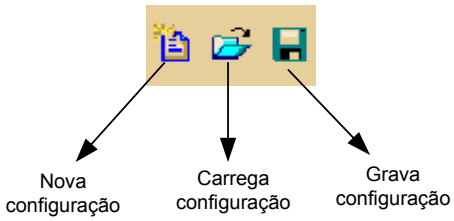

Fig. 3.14 – Manipulação de configurações

As operações a efectuar sobre as configurações encontram-se neste grupo e são básicamente três: 1- criar uma nova configuração; 2- Carregar uma configuração préviamente definida; 3- Gravar configuração existente.
Como se pode observar pela figura 3.14, os botões tentam ser sugestivos à operação associada, mantémdo-se dentro do standard das aplicações Windows.

Operações

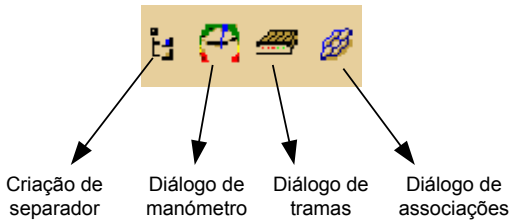

Fig. 3.15 - Operações

Este é sem dúvida o grupo mais importante da consola. Representa todas as operações fundamentais para criar uma configuração. A ordem com que os botões estão organizados tenta sugerir ao utilizador a sequência correcta de criação de configurações (Fig. 3.15). Assim, da esquerda para a direita, temos:

- um botão de criação de um separador,
- um botão para criação de um manómetro (através do qual são definidas todas as propriedades relevantes)
- um botão para o menu de tramas, através do qual serão definidas e editadas as tramas da configuração
- um botão para o diálogo que permite a associação trama/manómetro

Cada um dos diálogos será descrito em pormenor.

Comunicação

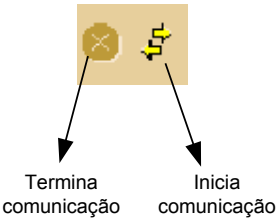

Fig. 3.16 - Comunicação

Neste grupo, o utilizador inicia/termina a comunicação com o veículo. Quando selecciona o botão de inicio de comunicação (Fig. 3.16), o estado da aplicação muda, inibindo os botões de modo a não permitir ao utilizador alerar a configuração até ai definida. Caso o utilizador pretenda alterar algum aspecto da configuração, terá de carregar no botão de paragem de comunicaçoes.

Setup

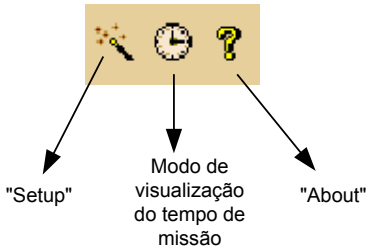

Fig. 3.17 - Setup

Neste grupo, o utilizador pode iniciar o diálogo de setup, própriamente dito, onde configura os endereços IP da consola. O modo de visualização do tempo de missão, relaciona-se com um controlo informativo situado na zona informativa da barra de comandos (será descrito na secção seguinte). Basicamente a sua função é alterar o modo como a informação referente ao tempo de missão é apresentada (alternando entre tempo total em segundos ou no format HH:MM:SS). O botão about apenas informa o utilizador da versão da consola.

Grupo Informativo

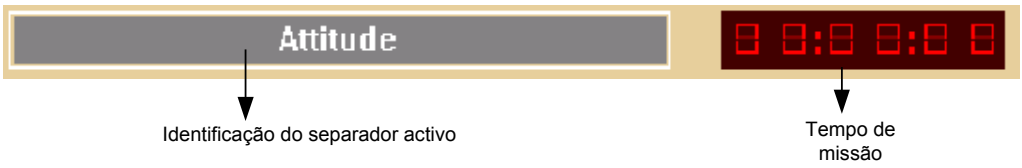

Fig. 3.18 – Grupo informativo

A Fig. 3.18 exemplifica o grupo informativo da barra de comandos. Esta é constituida por duas zonas: Identificação do separador activo e contador do tempo de missão, representado por um LED numérico. Como referido, o modo de visualização do tempo de missão pode-se alternar entre o format HH:MM:SS ou os segundos totais desde o inicio.

3.2.1.3 Manómetros

Os elementos fundamentais da interface são os manómetros. Estes são utilizados, tal como já foi referido, tanto para representar o estado do veículo, como também para o comandar sendo a interacção com estes realizada através do rato.

A sua interface (manómetros), apesar de algumas limitações, permite a configuração de uma grande variedade de parâmetros que vão desde a possibilidade de interacção com o rato até à grossura das agulhas, passando também pelo número de "escalas" a utilizar, gamas de medida, variação dos indicadores intermédios (*delta*), grossura das agulhas e cores entre muitos outros. Em suma, é permitida a configuração em geral de cada um dos tipos de manómetros, podendo desta forma adaptar-se às necessidades do veículo a monitorizar/comandar. A fig. 3.19 representa um manómetro do tipo linear e respecticvos diálogos de configuração.

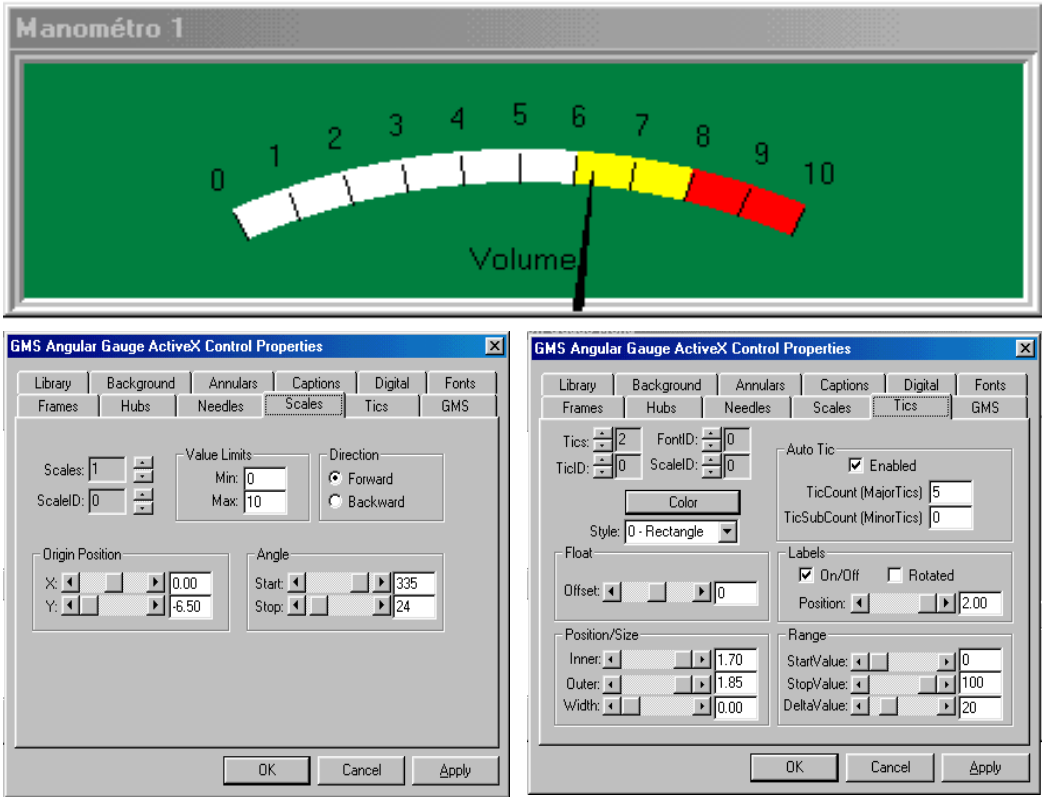

Fig. 3.19 – Exemplo do interface de configuração de um manómetro

3.2.2 Interface do diálogo de definição de tramas

A Fig. 3.20 orienta o utilizador relativamente aos tipos de acções disponíveis (criação, edição ou remoção de tramas), utilizando para isso como modo de operação, escolha forçada ou livre. Isto significa que em cada instante os botões de acção representam através da sua forma a possibilidade ou impossibilidade de poder ser efectuada a acção correspondente.

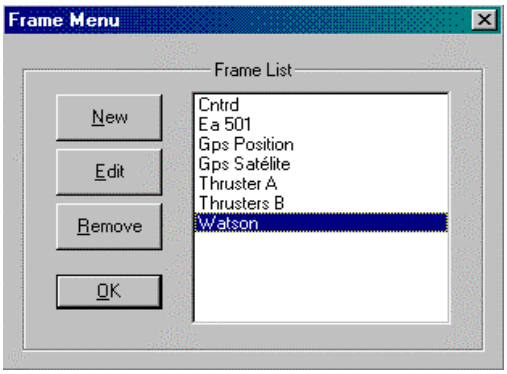

Fig. 3.20 - Diálogo de Tramas

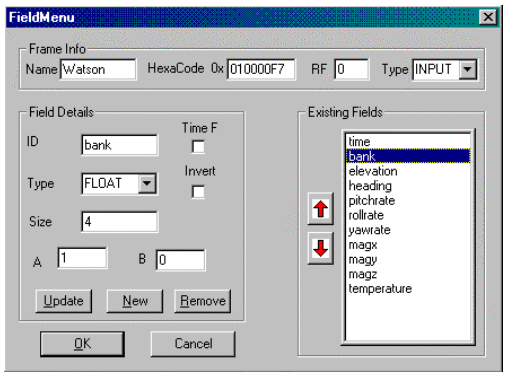

Fig. 3.21 - Configuração das tramas

Este modelo de operação melhora a navegação no sistema, ao mesmo tempo que evita inundar o utilizador com mensagens de erro. Por exemplo, no caso de não haver nenhuma trama definida não faz sentido poder editá-la ou remove-la. Nesse caso, os botões de acção *Edit* e *Remove* estarão indisponíveis.

No que diz respeito à criação da trama tentou-se criar um diálogo suave e de fácil interpretação para o utilizador (ver Fig. 21). Uma *List Box* do lado direito contém o registo de todos os campos já definidos para a trama na respectiva ordem. Ao seleccionar um campo, a sua informação surge no lado esquedo do diálogo, nomeadamente a informação referente ao nome do campo, tipo, tamanho (em bytes), parâmetros *a*, *b*, *invert, e TimeF*. Esta solução permite organizar, sem ocupar muito espaço de visualização, informação do mesmo tipo de uma forma lógica e estruturada, evitando mais uma vez neste caso ambiguidades.

Neste diálogo, à semelhança do que acontece com outros elementos da interface, utilizam-se valores estáticos para o caso dos campos não preenchidos e sem grande relevância no contexto de execução. Por exemplo, quando o utilizador selecciona um tipo para o campo, automaticamente o tamanho é preenchido com o seu valor de defeito.

3.2.3 Interface do diálogo de criação de manómetros

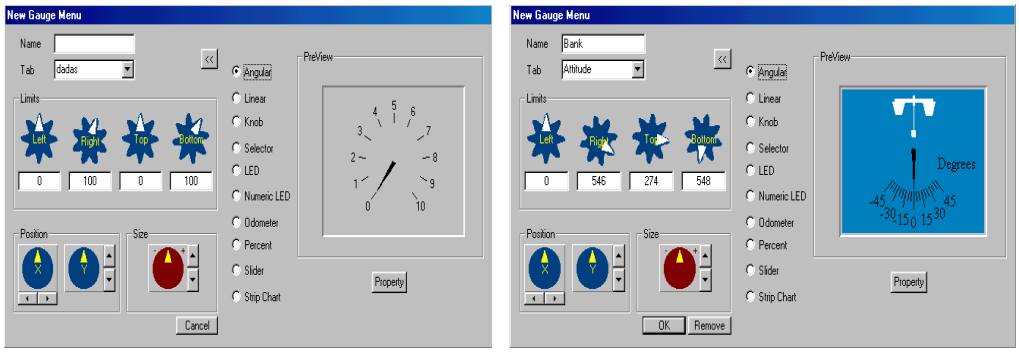

Fig. 3.22 – Diálogo usado para criar e configurar manómetros

Neste diálogo introduzimos uma lista de *Radio-Buttons* pois o número de opções é elevado e por outro lado, procuramos optimizar o espaço disponível. Introduzimos também a pré-visualização de maneira que o utilizador obtenha imediatamente resultados à medida que procede à configuração do manómetro que pretende criar. Como o processo de criação e configuração do manómetro envolve um grande número de parâmetros a definir, a possibilidade de erro é elevada, pelo que desta forma se evita que o utilizador tenha constantemente que repetir o processo. Utiliza-se mais uma vez a escolha forçada de forma a garantir a correcta execução do processo de configuração. Como exemplo citamos o botão *Ok* que apenas está disponível quando foi dado um nome ao manómetro e foi activado um *radio-button* seleccionando assim um manómetro.

3.2.4 Interface do diálogo das associações

Este diálogo resulta do exposto nas secções anteriores e da incorporação de um novo objecto: árvores de representação. A sua aplicação neste caso permite igualmente um mecanismo de visualização instantânea do estado dos objectos criados adaptada à própria estrutura de trama.

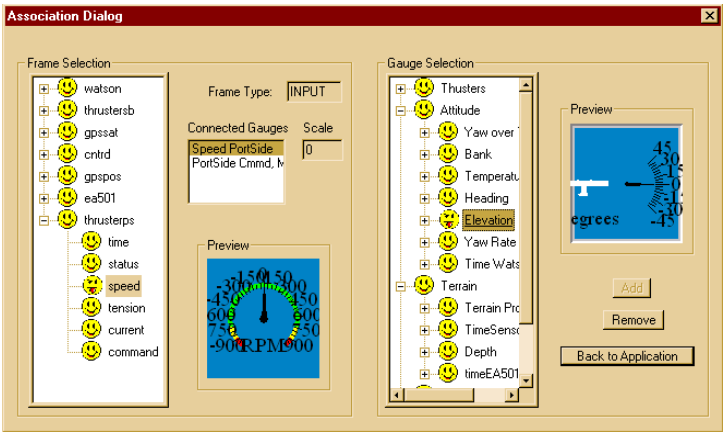

Fig. 3.23 – Diálogo para associação tramas/manómetros

Assim, o utilizador tem à sua disposição duas árvores com a informação relevante para a associação trama/manómetro (Fig.3.23). A árvore da esquerda representa todas as tramas existentes na configuração. Expandindo esta árvore, surgem os campos para cada uma das tramas. Ao seleccionar um campo, automaticamente aparece na janela "*Connected Gauges"* a lista de manómetros associados (se existirem). Ao seleccionar um deste manómetros, a informação da escala associada é actualizada na janela "*scale*", ao mesmo tempo que aparece uma pré-visualização na janela "*Preview*". Deste modo o utilizador pode facilmente visualizar os manómetros associados aos campos das tramas (e respectivas escalas).

Do lado direito da caixa de diálogo encontra-se uma árvore que representa todos os separadores existentes. Expandindo esta árvore surgem os manómetros para cada um dos separadores e, no nível seguinte, as escalas que cada um possui. Ao seleccionar um manómetro nesta árvore, automaticamente surge uma pré-visualização do mesmo na sua janela "*Preview*".

Quando existe um campo de trama seleccionado e uma escala de manómetro, o botão "*Add*" e "*Remove*" ficam activos pois estão reunidas as condições para se proceder a uma associação/desassociação entre os elementos seleccionados. Refira-se que se o manómetro já estiver associado a outro campo surgem caixas de diálogo a informar o utilizador, esperando que este confirme a acção.

3.2.5 Interface em geral

Os requisitos desta aplicação impõem fundamentalmente a consideração da manipulação directa e de menus como estilos de diálogo. Não foram considerados no estudo para a interface, outros estilos de diálogo tais como formulários, pergunta-resposta, linguagens de comandos (substituído pelo sistema aqui descrito) e teclas de função (se bem que parcialmente incorporado na aplicação).

A implementação do sistema de interface e respectiva integração no núcleo funcional implicou o desenvolvimento de um protótipo de interface que foi refinado sucessivamente até que se reflectissem todos os princípios e opções acima descritos. Estas opções, surgem naturalmente a partir da identificação dos estilos de diálogo mais adequados a adoptar nas diversas facetas do

processo de comunicação dependente do fluxo/sequência de operações a implementar e do estado actual do sistema.

Seguiu-se o desenvolvimento do núcleo funcional da consola através da definição e implementação do mecanismo de envio/recepção de tramas e definição das estruturas necessárias ao suporte das tramas, manómetros e associações. Posteriormente foi realizada a integração do núcleo funcional com a interface e alguns melhoramentos da mesma, eliminando assim algumas falhas detectadas.

Achou-se importante minimizar os recursos de aprendizagem e de conhecimento – recursos cognitivos – de modo a não hostilizar o utilizador com a obrigação de um estudo antecipado do funcionamento da aplicação. No fundo tiveram-se como princípios gerais a simplicidade, a manipulação directa, o controlo constante das operações, a flexibilidade e capacidade de reacção, rapidez de execução, protecção contra erros ou anomalias decorrentes do processo de execução, a facilidade de aprendizagem bem como de utilização.

Do ponto de vista da ambientação, do bem estar do utilizador e da sua motivação houve a preocupação em fornecer realimentação adequada através do desenho destacado dos ícones, e minimizar o movimento dos olhos através da redução da necessidade de leitura/procura e do agrupamento dos ícones representativos das acções na mesma área de visualização. Procurou-se evitar problemas de semântica a um nível demasiado elevado pelo que se tentou que as formas e imagens escolhidas fossem simples e sugestivas. Evitou-se também, como já foi referido, realimentações irrelevantes ou ambíguas.

As relações com as unidades E/S – Entrada/Saída, são estabelecidas pelo núcleo da MFC que integra já o suporte para os dispositivos de interacção referidos.

Relativamente aos menus, estes não se mostraram especialmente práticos. Apesar de auto-explicatvos, fácil gestão de erros e visibilidade de novas funções, traduzem uma navegação ineficiente, pouca flexibilidade, um número limitado de opções e utiliza a área de visualização, escondendo assim a área de desenho. Não motivam nem ambientam o utilizador. Implicam uma habituação às teclas para uma rápida actuação. Neste aspecto, a manipulação directa apresentou-se como estilo de diálogo mais apropriado. Apesar disso foi implementado um menu com um modo de activação *Pull-Down.*

Foi também desenvolvido o mecanismo de *Arrastar e Largar*, em inglês *Drag & Drop*, tentando dessa forma aproximar-nos do processo de manipulação de objectos do sistema operativo *Windows* a que, a maioria dos possíveis utilizadores da consola, estão habituados (Fig. 3.24).

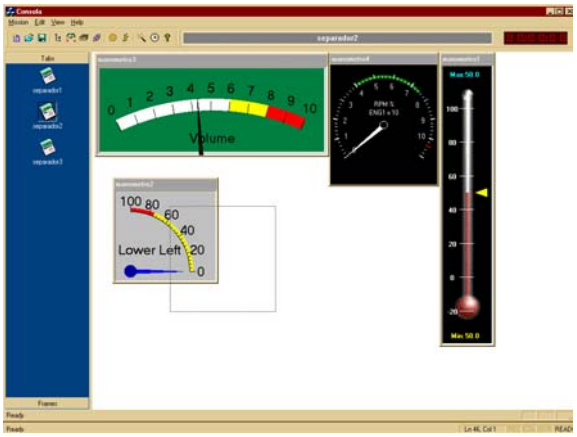

Fig. 3.24 – Exemplo de *Drag and Drop*

Os diálogos de mensagens foram também alvo de atenção: reduziu-se a dimensão das mensagens tornando-as específicas, incorporaram-se ícones para chamar a atenção utilizando cores diferentes conforme a situação: amarelo no caso de mensagens de aviso que notificam perigo, e Azul no caso de esclarecimentos. Do ponto de vista da navegação a aplicação é simples e eficiente, apesar do processo de configuração poder ser bastante repetitivo se estiverem envolvidas muitas variáveis a comandar ou monitorizar.

3.3 Estruturas de Dados

A ideia base é permitir ao utilizador a definição do formato das tramas, manómetros e associações entre ambos. De seguida tentaremos fazer uma abordagem descendente, começando por descrever o funcionamento conceptual da aplicação (ignorando detalhes) para depois começarmos a refinar a explicação e a incorporar pormenores relevantes ao correcto funcionamento da mesma.

As tramas são constituídas pelo menos por dois campos obrigatórios (um para o código da trama, outro para a hora em que foi criada). Os restantes campos são em número variável, consoante as necessidades pretendidas (Fig. 3.25). Os tipos e o formato de cada campo serão explicados posteriormente.

A aplicação terá um conjunto de manómetros criados pelo utilizador, agrupados em separadores (também criados pelo utilizador, consoante ache oportuno). Estes manómetros podem ser de vários tipos e possuir várias escalas, isto é, um único manómetro pode conter várias marcas e escalas (por exemplo um relógio tem ponteiros para as horas, minutos e segundos). Cada uma destas escalas estará, caso o utilizador o defina, associada a um campo de uma trama.

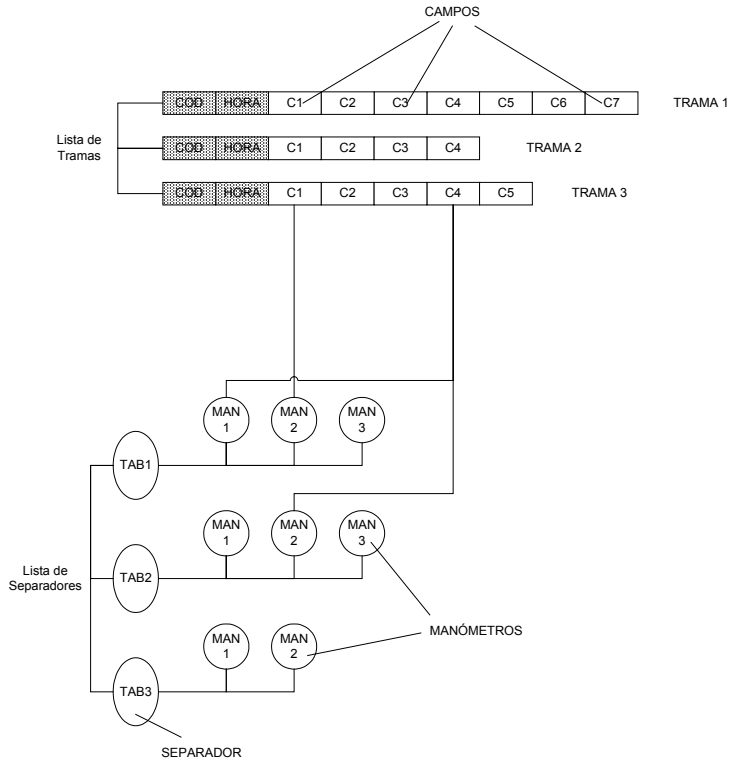

Fig. 3.25 – Panorâmica conceptual das estruturas usadas

São estas associações entre tramas e escalas de manómetros que constituem o núcleo do funcionamento da aplicação.

Uma vez recebida uma trama de comunicação (trama do tipo INPUT), após interpretar o código da trama e verificar que é uma trama existente na lista de tramas definidas pelo utilizador, a aplicação irá ler os campos da referida trama e, caso o campo possua uma lista de manómetros associados, será feito o refrescamento do(s) mesmo(s).

As tramas do tipo OUTPUT terão um funcionamento diferente. Estas tramas serão tramas a enviar ao veículo com vista a executar uma ordem. Neste caso, se a trama tiver um manómetro associado, quando o utilizador altera o valor desse manómetro, imediatamente é identificada a trama associada e lidos os valores dos manómetros associados aos campos da trama. À medida que esses valores vão sendo lidos, a trama é formada. Por fim, acrescenta-se o cabeçalho (código da trama e registo temporal). Se o campo de uma trama de OUTPUT não estiver associado a nenhum manómetro o seu valor será 0.

Esta é a ideia conceptual do funcionamento da nossa aplicação, no que se refere ao suporte de dados. A principal dificuldade reside no facto de tudo isto ter de ser o mais genérico possível. Embora perdendo algumas potencialidades inerentes ao desenho específico para apenas um objectivo, a aplicação ganha na potencialidade que tem ao permitir definir, dentro de alguns limites, qualquer formato de tramas e associar a um tipo de manómetros, permitindo assim criar configurações de monitorização e actuação para qualquer veículo.

Cada campo de uma dada trama pode estar associado a zero, um ou mais manómetros, dependendo da vontade do utilizador. Dado que podem existir tramas de INPUT e de OUTPUT a única restrição nesta associação é que o manómetro seja consistente com o tipo de trama associado, isto é, um manómetro não poderá estar simultaneamente associado a uma trama de INPUT e de OUTPUT.

Resumindo, classificamos os manómetros de OUTPUT como sendo aqueles em que o utilizador não pode alterar o seu estado e os de INPUT como sendo manómetros que o permitem. As tramas associadas serão de INPUT e OUTPUT respectivamente. Classificamos as tramas de INPUT como sendo aquelas que a aplicação recebe e as de OUTPUT como sendo as criadas e enviadas para o veículo.

Uma possível alteração ao funcionamento da aplicação seria permitir um manómetro com várias escalas ter associado a cada uma das suas diferentes escalas, tramas de tipo diferentes. Nesse caso, a classificação não seria feita ao nível do manómetro mas sim ao nível da escala do manómetro. (obviamente que a mesma escala nunca poderia estar associada a duas tramas de tipos diferentes). Este aspecto não foi considerado na implementação de forma a não complicar o processo de configuração, tal como nos foi recomendado.

O passo seguinte consistiu em escolher estruturas de dados que traduzissem eficientemente o raciocínio conceptual descrito anteriormente. Deste modo iremos explicar e justificar as estrutura de dados escolhidas para representar os manómetros, as tramas e as associações entre ambos.

3.3.1. Estruturas dos manómetros

Um dos requisitos essenciais era a generalidade pretendida para esta aplicação. Se se tratasse de fazer um consola para um veículo específico, cujas tramas são conhecidas, poderíamos simplificar imenso todo o processo. No entanto não é esse o objectivo. A aplicação terá de estar desenhada de forma a poder trabalhar com qualquer configuração definida pelo utilizador. Deste modo, não se conhecem à partida os manómetros que irão ser usados.

A solução foi permitir ao utilizador usar dez tipos base de manómetros nas suas configurações. A partir destes tipos base, o utilizador personaliza a configuração do manómetro. Assim, existem os seguintes tipos base:

1. *Manómetros Angulares*. Um dos mais usados em interfaces técnicas
2. *Knob* . (man. Arredondados). Ideal para fazer botões arredondados
3. *Selectores*. Ideal para fazer selectores
4. *LED*. Criação de LEDs usados em interfaces técnicas
5. *Manómetros Lineares*. Óptimo para interfaces técnicas.
6. *Odómetro*. Objecto similar aos odómetros dos automóveis
7. *Manómetros de percentagem*. Adequado para criação de gráficos de progresso
8. *Manómetros deslizantes*. Usado em interfaces técnicas.
9. *Manómetros 'Strip Chart'*. Para criar gráficos de uma ou mais variáveis
10. *Manómetros numéricos*. São LEDs numéricos óptimos para display de valores

No entanto, devido à generalidade pretendida, pretende-se ter um objecto global que passamos a designar de manómetro, sobre a qual todas as operações serão efectuadas. Recorrendo ao polimorfismo de inclusão disponibilizado pelo paradigma de programação orientado a objectos (C++), consegue-se o objectivo pretendido, isto é, trabalhar com um manómetro seja ele de que tipo base for. De seguida apresenta-se um gráfico sugestivo:

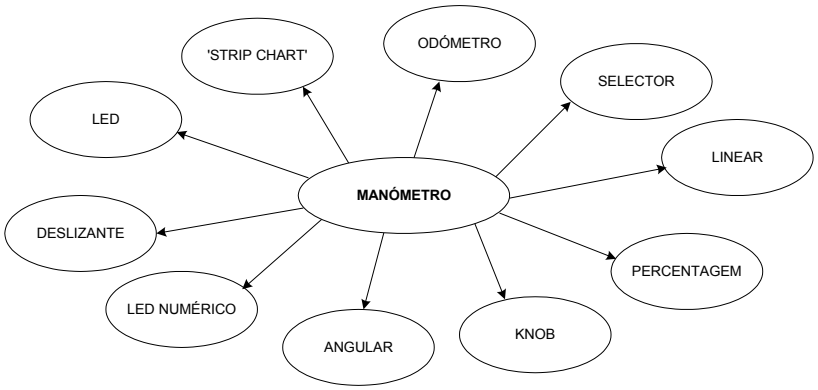

Fig. 3.26 – Manómetros

Todos os métodos relevantes a usar são definidos como virtuais na classe manómetro, estando a sua implementação encapsulada nas subclasses respectivas, atribuindo assim um comportamento específico a cada um dos manómetros.

O utilizador, ao criar uma configuração, define o tipo de manómetro desejado e as suas propriedades (nome, coordenadas, tipo). Após criar o manómetro, este é inserido no separador desejado. Assim, cada separador irá conter uma lista dos manómetros que lhe estão associados. A sua estrutura é extremamente simples (Fig 3.27), consistindo apenas numa referência a uma lista de manómetros associados e de três variáveis: uma com o nome do separador, com o

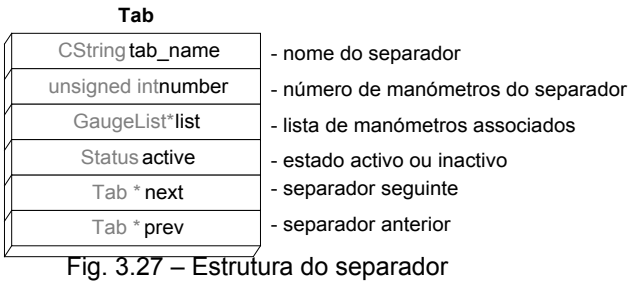

Fig. 3.27 – Estrutura do separador

número de manómetros associados e com o estado do separador (activo ou inactivo). Finalmente têm uma referência para o separador seguinte e para o anterior (caso existam). A função da variável de estado é indicar à aplicação qual o separador activo que é necessário "refrescar" no

ecrã. Portanto, no funcionamento normal da aplicação irão ser criados vários separadores, cada um dos quais contendo manómetros, criando assim uma lista.

Vamos agora explicar com mais detalhe o objecto, anteriormente designado manómetro. Para este tipo de objecto é definida uma classe – a classe *CGauge* .
Como se pode observar pela figura anterior, o separador possui um campo que é uma lista de manómetros (o campo list) . Esta lista é representada por uma estrutura simples,

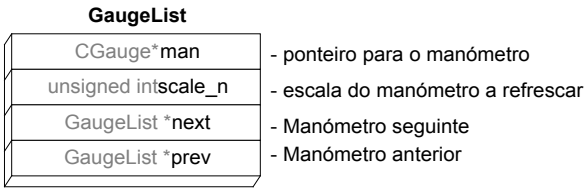

Fig. 3.28 – Estrutura da lista de manómetros

A lista de manómetros não é mais que um ponteiro para um objecto da classe CGauge (que é a classe representativa dos manómetros), uma variável inteira com o número da escala a refrescar (esta função será explicada mais à frente quando abordarmos as associações entre tramas e manómetros) e dois ponteiros, um para o manómetro anterior, outro para o seguinte.

Como se observa na Fig 3.28, a variável *man* representa um ponteiro para um manómetro, seja ele de que tipo for. Assim, quer seja um manómetro angular, quer seja linear, quer seja de um outro tipo base qualquer, será sempre um manómetro, pelo que se consegue obter uma abstracção importante no tratamento dos dados. Esta funcionalidade é obtida pelo polimorfismo de inclusão comum nas linguagens orientadas a objectos tal como é o Visual C++.

Vamos agora descrever a classe CGauge, usada para representar um objecto do tipo manómetro.

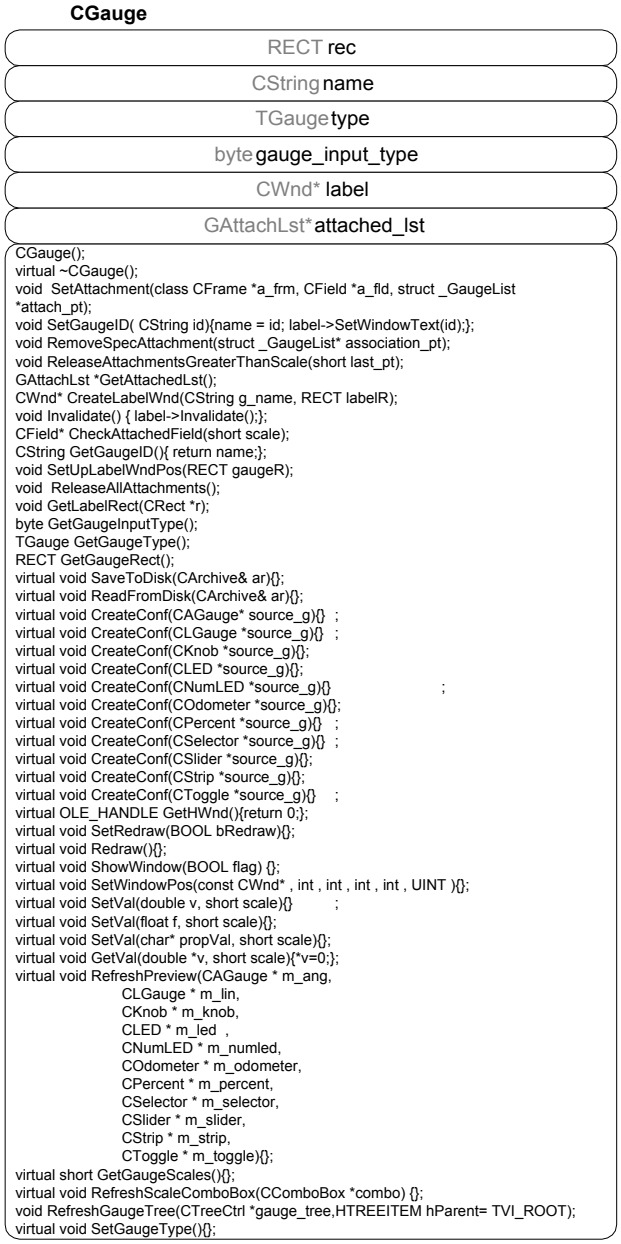

Fig. 3.29 – Estrutura do manómetro

Esta classe possui seis variáveis internas, que passamos a descrever:

- *rec* do tipo RECT, representa os limites da janela do manómetro
- *name* do tipo CString, nome do manómetro
- *type* do tipo TGauge, tipo base do manómetro (Angular, Linear , ...)
- *gauge_input_type* do tipo byte, interacção com o manómetro (INPUT/OUTPUT)
- *label* do tipo CWnd, janela que contem o próprio manómetro
- *attached_lst* do tipo GAttachLst, representa as associações em que o manómetro participa

Foram também definidos vários métodos de manipulação destas variáveis. Por outro lado possui muitos métodos virtuais, pois o tratamento e manipulação da informação refere-se sempre a um manómetro. O polimorfismo de inclusão encarrega-se de associar o objecto manómetro (genérico) ao objecto manómetro base e invoca o método correspondente. Daí a necessidade de definir alguns métodos virtuais nesta classe. A implementação destes métodos encontra-se na definição da classe base do objecto em questão.

Resumindo, a lista de manómetros suporta objectos do tipo manómetro genérico (CGauge), pelo que operações sobre esse objecto têm de estar definidas na sua classe. Como os objectos podem ser de vários tipos base (Linear, Angular, LED....), ao invocar o método X, definido na classe do manómetro genérico, o polimorfismo encarrega-se de verificar que esse método é um método virtual, indo de seguida ao objecto da classe base, executar o código na sua implementação.
Este processo ilustra-se facilmente através de um exemplo.

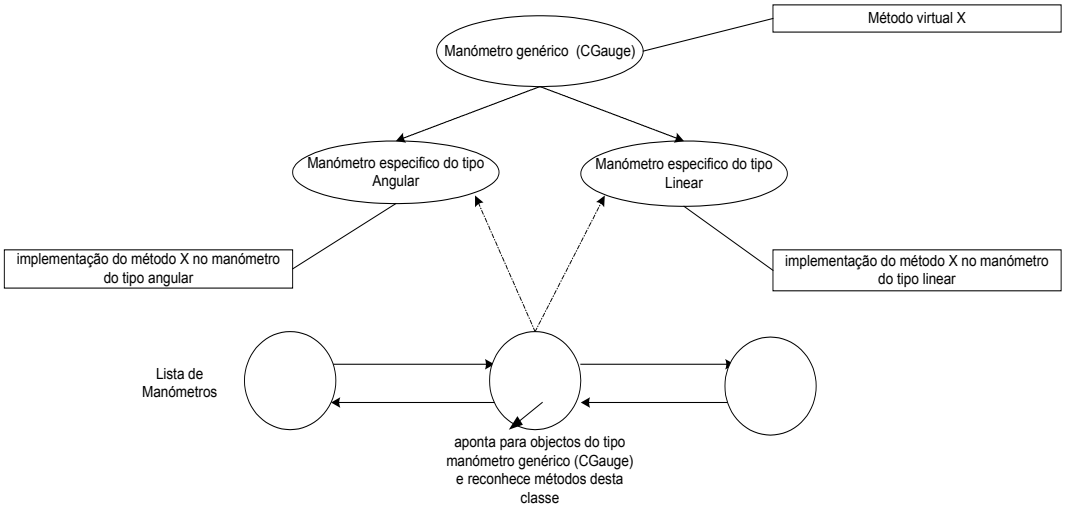

Fig. 3.30 – Polimorfismo

Como se observa através da Fig. 3.30, a lista de manómetros lida com objectos do tipo manómetro genérico. Como qualquer manómetro de um tipo específico é um manómetro do tipo genérico (pois as respectivas classes são subclasses de CGauge) a lista irá conter manómetros específicos. No exemplo, tanto podia ser um manómetro angular, como linear. Ao invocar o método virtual X da classe CGauge, automaticamente é reconhecido o tipo base associado ao objecto e invocado o método respectivo.

Foi escolhida uma estrutura do tipo lista pois considerámos que o número de manómetros existentes em cada separador não iria ser elevado de modo a justificar outro tipo de estrutura mais complexo. Quando o utilizador muda o separador activo, é necessário inibir as janelas de todos os manómetros existentes no separador antigo e activar as janelas dos manómetros existentes no novo separador. Uma lista permite aceder aos manómetros existentes nos separadores de forma eficaz e rápida. De seguida apresentam-se os métodos virtuais definidos na classe manómetro. Estes métodos constituem as acções mais importantes a realizar sobre os manómetros existentes nas configurações.

Nome	**Descrição**
`void SaveToDisk(CArchive)` `void ReadFromDisk(CArchive&)`	*Gravação do manómetro no disco* *Leitura do manómetro a partir do disco*
`void CreateConf(CAGauge* )` `void CreateConf(CLGauge*)` `void CreateConf(CKnob*)` `void CreateConf(CLED* )` `void CreateConf(CNumLED*)` `void CreateConf(CCOdometer*)` `void CreateConf(CPercent*)` `void CreateConf(CSelector*)` `void CreateConf(CSlider*)` `void CreateConf(CStrip*)` `void CreateConf(CToggle*)`	*As funções CreateConf servem para copiar uma configuração de um manómetro para outro. Por exemplo, se o argumento for do tipo base CLED, o manómetro adquire o formato do tipo LED*
`OLE_HANDLE GetHWnd()`	*Devolve o handler para a janela que contém o manómetro*
`void SetRedraw(BOOL)` `void Redraw()` `void ShowWindow(BOOL)` `void SetWindowPos(const CWnd* , int , int , int , int , UINT )`	*Altera a flag de refrescamento do manómetro* *Refresca o manómetro* *Esconde / mostra a janela do manómetro* *Altera a localização da janela do manómetro*
`void SetVal(double, short )` `void SetVal(float, short )` `void SetVal(char*, short )` `void GetVal(double *, short )`	*SetVal actualiza uma escala do manómetro com um valor* *Devolve o valor do manómetro*
`void RefreshPreview(CAGauge*, CLGauge*, CKnob*, CLED*, CNumLED*, CCOdometer*, CPercent*, CSelector*, CSlider*, CStrip*, CToggle*)`	*Refresca as janelas dos manómetros em algumas caixas de diálogo*
`void RefreshScaleComboBox(CComboBox*)` `void RefreshGaugeTree(CTreeCtrl*, HTREEITEM )` `short GetGaugeScales()` `void SetGaugeType()`	*Preenche uma combo box com as escalas* *Preenche um controlo do tipo "árvore" com as escalas* *Devolve o número de escalas* *Altera o tipo do manómetro*

Fig. 3.31 – Métodos de CGauge

3.3.2 Estruturas das tramas

Vamos agora falar um pouco das tramas e sua constituição. Em primeiro lugar começamos por referir que as tramas, ao ser definidas pelo utilizador, irão ficar armazenadas numa tabela de Hash, de modo a que o durante o processo "offline" de configuração em que as tramas podem ser redefinidas o acesso às mesmas seja feito de forma mais rápida. (Fig. 3.32)

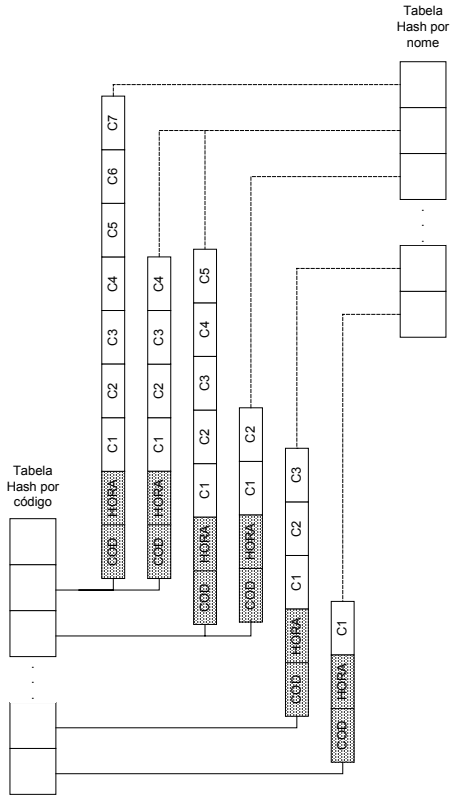

Fig. 3.32 - Estrutura conceptual da organização das tramas .
Como se pode observar, as tabelas de Hash terão a organizações diferentes, pois aplicam funções diferentes para determinar os índices. Quando uma trama é removida/criada/actualizada, as listas contidas nas tabelas serão actualizadas simultaneâmente

Esta tabela de HASH usa o nome da trama para determinar o índice de armazenamento, com vista a facilitar a interacção ao utilizador. Existe ainda uma segunda tabela de HASH, que referencia também as tramas. No entanto esta segunda tabela não usa o nome, mas sim o código hexadecimal definido pelo utilizador.

Este código é único para cada trama. A finalidade desta segunda tabela é acelerar o processo de identificação da trama, durante o funcionamento da aplicação. Isto porque todas as tramas serão identificadas por um código.

Ao receber uma trama, a aplicação descodifica o código da trama referida. De seguida consulta a tabela de códigos para ver se existe alguma trama definida com aquele código. Em caso afirmativo, lê o conteúdo dos campos da referida trama (este processo será descrito em mais profundidade). Devido ao facto de ser necessário identificar tramas e aceder à sua estrutura durante o funcionamento da aplicação, pretende-se diminuir o tempo de procura da trama, pelo que a tabela de HASH apresenta-se como uma boa solução, ao invés de uma única lista com todas as tramas definidas.

Não esquecer que durante o funcionamento do veículo a aplicação estará constantemente a receber tramas sendo necessário actualizar os valores dos manómetros associados aos campos da trama. Daí que seja fundamental optimizar os tempos de acesso aos dados.

Claro que no processo de definição das tramas, sempre que o utilizador apaga/cria/altera tramas, as duas tabelas de HASH são correctamente actualizadas. Estas tabelas de HASH, possuem listas de tramas, cuja estrutura se encontra definida de seguida:

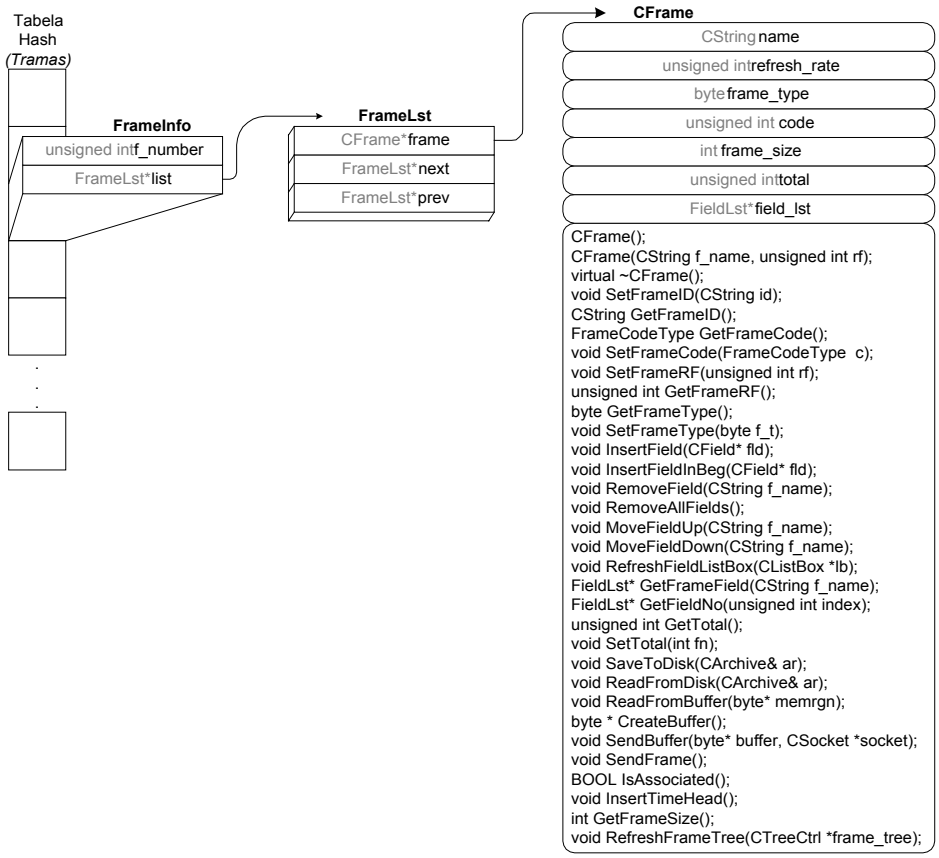

Fig. 3.33 - Trama

A trama é representada pela classe CFrame,

Variáveis internas:

- *name* . Nome associado à trama
- *refresh_rate*. Taxa de refrescamento da trama
- *frame_type*. Tipo da trama (INPUT/OUTPUT)
- *code*. Código hexadecimal da trama (pelo qual é inserida na tab. Hash)
- *frame_size.* Tamanho, em bytes, da trama (usado na leitura da trama)
- *total.* Número total de campos da trama

- *field_lst.* Lista de campos da trama. A ordem dos campos na lista é relevante pois será por esta ordem que a aplicação irá ler a informação referente a cada um dos campos. O processo de leitura e escrita das tramas será abordado em pormenor.

Métodos:

Nome	**Descrição**
`CFrame()` `CFrame(CString, unsigned int)`	*Constructor*
`virtual ~CFrame()`	*Destrutor*
`void SetFrameID(CString)` `CString GetFrameID()`	*Define identificação da trama* *Devolve a identificação da trama*
`void SetFrameCode( FrameCodeType )` `FrameCodeType GetFrameCode()`	*Define o código hexadecimal da trama* *Devolve código hexadecimal da trama*
`void SetFrameRF(unsigned int)` `unsigned int GetFrameRF()`	*Define a taxa de refrescamento da trama* *Devolve a taxa de refrescamento da trama*
`byte GetFrameType()` `void SetFrameType( byte )`	*Devolve o tipo da trama* *Define o tipo da trama*
`unsigned int GetTotal()` `void SetTotal( int )`	*Devolve o total de campos da trama* *Altera o total de campos da trama*
`int GetFrameSize()`	*Devolve o tamanho da trama*
`void InsertField(CField*)`	*Insere campo na trama (insere no fim da lista)*
`void InsertFieldInBeg(CField*)`	*Insere campo no inicio da lista*
`void RemoveField(CString)`	*Remove campo da trama, por nome*
`void RemoveAllFields()`	*Remove todos os campos da trama*
`void MoveFieldUp( CString )`	*Avança campo uma posição na lista*
`void MoveFieldDown(CString)`	*Recua campo uma posição na lista*
`void RefreshFieldListBox(CListBox*)`	*Refresca uma "list box" com os nomes dos campos*
`FieldLst* GetFrameField(CString)`	*Devolve um campo da trama (por nome)*
`FieldLst* GetFieldNo(unsigned int)`	*Devolve um campo da trama (por posição na lista)*
`void SaveToDisk(CArchive& )`	*Grava informação da trama trama no disco*
`void ReadFromDisk(CArchive& )`	*Recolhe informação da trama do disco*
`void ReadFromBuffer(byte*)`	*Lê trama de um buffer*
`byte* CreateBuffer()`	*Cria um buffer com a info da trama*
`void SendBuffer(byte*, CSocket*)`	*Envia buffer para o socket*
`void SendFrame()`	*Verifica socket e envia buffer para lá*
`BOOL IsAssociated()`	*Verifica se a trama está associada a manómetros*
`void InsertTimeHead();`	*Insere "selo" temporal no cabeçalho da trama*
`voidRefreshFrameTree(CTreeCtrl *frame_tree);`	*Refresca um comando do tipo árvore com o conteúdo da trama*

Fig. 3.34 – Métodos de CFrame

Como se pôde observar, as tramas possuem uma lista de campos, lista essa definida pelo utilizador. A posição dos campos na lista é extremamente importante e deve ser salvaguardada, pois a informação quando for lida será feito através de um buffer que é decomposto pela ordem

dos campos da lista. Como iremos ver já a seguir, os campos possuem um atributo que representa o seu tamanho em bytes e será precisamente esse tamanho que é lido do buffer inicial. Desta forma, podemos ler qualquer tipo de trama definida pelo utilizador. Este processo de leitura e criação de tramas será descrito em secções posteriores. Analisemos a estrutura que representa a lista de campos e o próprio campo,

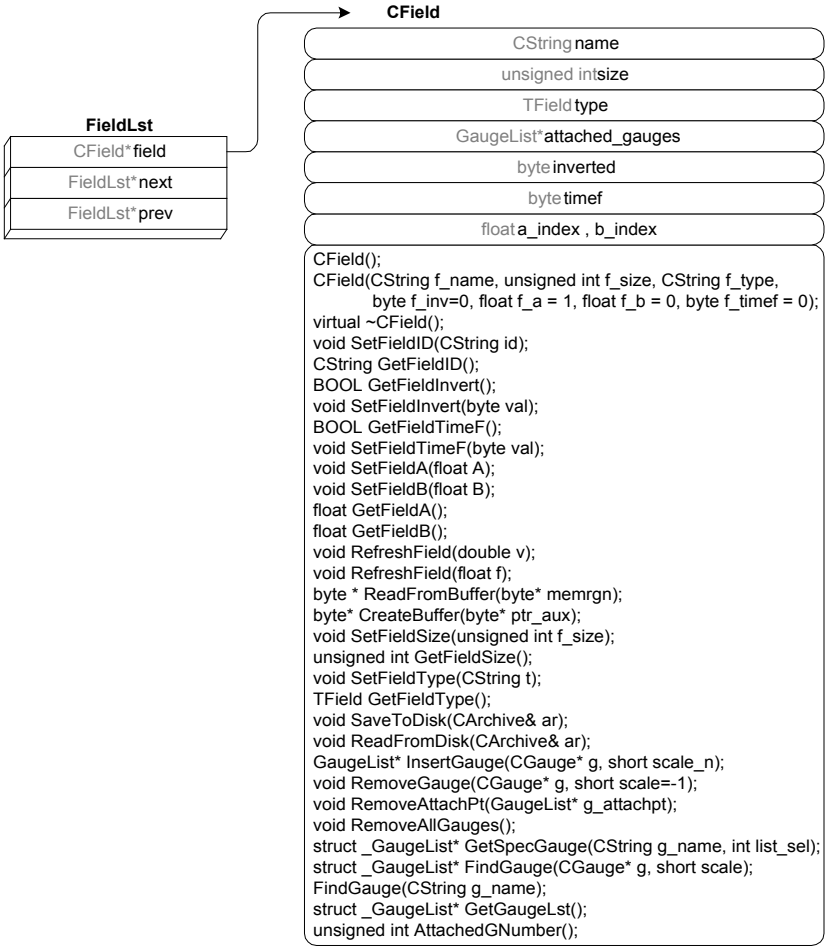

Fig. 3.35 – Estrutura do campo da trama - CField

A campo da trama é representada pela classe CField.

Variáveis internas:

- *name* Nome associado ao campo
- *size* Tamanho do campo em bytes
- *attached_gauges* Lista de manómetros associados ao campo. Esta variáveis será explicada melhor durante a descrição do processo de associação trama/manómetro.
- *type* Tipo de dados base do campo (inteiro, byte, float...)

- *inverted* Flag que indica se é necessário inverter os dados durante a leitura/criação da trama. O hardware existente no veículo pode processar dados de forma diferente do H/W onde corre a aplicação. Ex: Motorola/Intel.
- *timef* Flag que indica a necessidade de processar um campo do tipo temporal. Será aplicada uma função que devolve o tempo formatado para apresentação em LEDs numéricos
- *a_index; b_index* Parâmetros aplicados ao valor lido sob a forma aX+b. Por vezes é necessário aplicar correcções aos valores lidos da trama, para poder alterar escalas durante, por exemplo, a visualização de gráficos

Métodos:

Nome	**Descrição**
`CField();` `CField(CString, unsigned int, CString, byte, float, float, byte);` `virtual ~CField();`	*Construtor* *Destrutor*
`void SetFieldID(CString);` `CString GetFieldID();`	*Define a identificação do campo* *Devolve a identificação do campo*
`BOOL GetFieldInvert();` `void SetFieldInvert(byte);`	*Devolva a flag de inversão dos dados* *Define a falg de inversão dos dados*
`BOOL GetFieldTimeF();` `void SetFieldTimeF(byte);`	*Devolve o valor da flag da função tempo* *Define o valor da flag da função tempo*
`void SetFieldA(float);` `void SetFieldB(float);` `float GetFieldA();` `float GetFieldB();`	*Manipulação das variáveis a e b* *(aX +b)*
`void SetFieldSize(unsigned int);` `unsigned int GetFieldSize();`	*Define o tamanho do campo (em bytes)* *Devolve o tamanho do campo*
`void SetFieldType(CString);` `TField GetFieldType();`	*Define o tipo do campo* *Devolve o tipo do campo*
`void RefreshField(double);` `void RefreshField(float);`	*Refresca todos os manómetros associados ao campo.*
`byte * ReadFromBuffer(byte*);`	*Lê a trama de um buffer e actualiza os campos*
`byte* CreateBuffer(byte*);`	*Cria um buffer com o valor dos campos da trama*
`void SaveToDisk(CArchive&);` `void ReadFromDisk(CArchive& );`	*Guarda a informação da trama no disco* *Recupera a informação da trama do disco*
`GaugeList* InsertGauge(CGauge*,short)` `void RemoveGauge(CGauge* , short )` `void RemoveAllGauges();` `void RemoveAttachPt(GaugeList*)` `GaugeList* GetSpecGauge(CString, int)` `GaugeList* FindGauge(CGauge*,short)` `GaugeList* GetGaugeLst()` `unsigned int AttachedGNumber();`	*Funções de manipulação das associações entre o campo e os manómetros associados*

Fig. 3.36 – Métodos de CField

3.3.3. Estruturas das associações trama/manómetro

Nesta secção iremos descrever as estruturas de dados usadas para representar as associações entre tramas e manómetros.Como vimos atrás, foi deixado espaço nas estruturas dos manómetros e dos campos das tramas para que se possa arranjar um processo de ligação entre estes, quando o utilizador os decide associar. O campo da trama, caso esteja associado a algum(s) manómetro(s), irá ter uma lista com todos os manómetros relacionados.

Por seu turno, cada um dos manómetros que esteja associado irá ter uma referência para o campo, respectiva trama e para uma estrutura, chamada ponto de associação, que indica qual a escala do manómetro associada. Esta estrutura é necessária pois um manómetro que tenha mais do que uma escala, poderá estar associado a mais do que um campo. Deste modo poderemos distinguir qual a escala do manómetro a refrescar quando for caso disso. Mais uma vez, iremos recorrer a uma representação gráfica com vista a ilustrar o que foi descrito

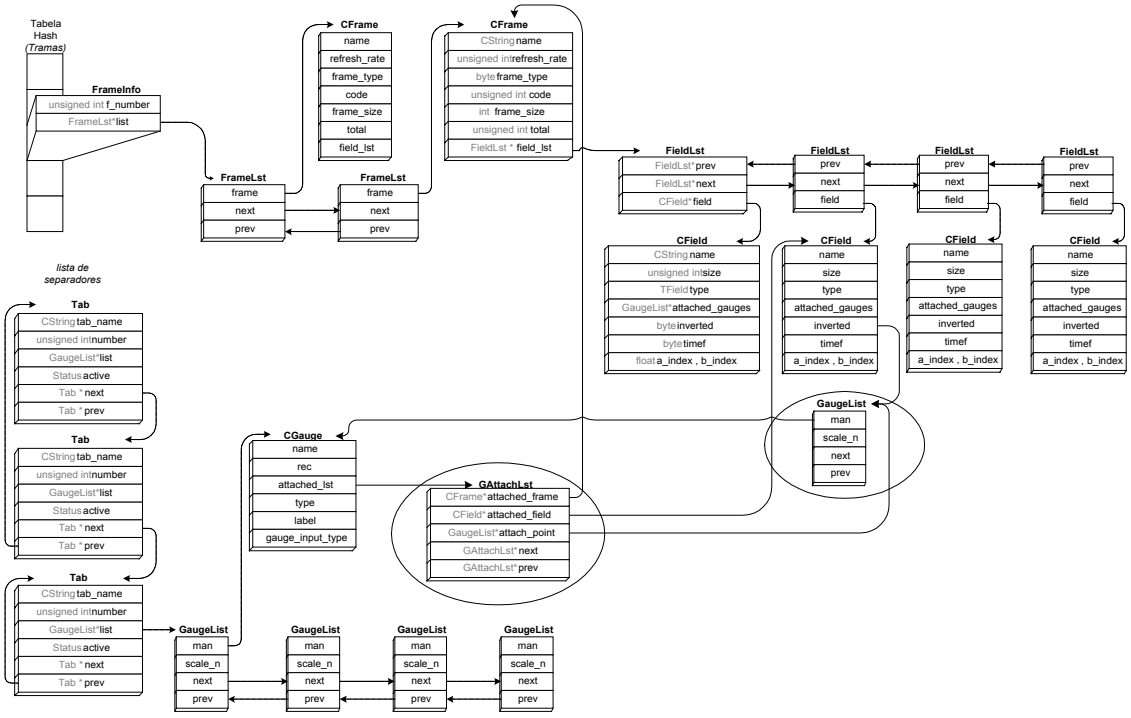

Fig. 3.37 – Estruturas criadas numa associação trama/manómetro

Neste exemplo, foi criada uma associação entre um manómetro existente no terceiro separador e o segundo campo de uma trama. Como se pode observar na estrutura do campo, a lista de manómetros associados é actualizada ficando a referenciar uma estrutura que indica qual o manómetro referenciado e a escala a actualizar. Por seu turno, no objecto manómetro (classe

CGauge) é também actualizada a lista de dependências, com a trama, o campo e o ponto de inserção (estrutura *GAttachLst*).

Se o manómetro tivesse mais campos associados (a escalas diferentes), o ponteiro representado por *next* da estrutura *GAttachLst*, referenciaria uma nova estrutura *GAttachLst* com a informação da nova referência. Por seu turno o mesmo se passa para o campo da trama. O campo pode perfeitamente estar associado a mais do que um manómetro, pelo que a lista representada por *attached_gauges* em *CField*, (que é uma estrutura representada por G*augeList)* seria actualizada, ficando o campo *next* de *GaugeList* referenciar o manómetro seguinte.

O processo repete-se à medida que se acrescentam associações. Sempre que são removidas ou alteradas, todos estes ponteiros são destruídos e as referências actualizadas.

3.3.4 Exemplo

Pensamos ser sugestivo descrever através de imagens a criação de uma trama, manómetro e respectiva associação. Vamos exemplificar, o que cada acção do utilizador gera em termos de estruturas de dados. Quando se inicia a aplicação todas as estruturas de dados estão vazias. (tabelas de HASH e a lista de separadores). Em primeiro lugar vamos definir um separador onde possamos inserir um manómetro (ver Fig. 3.38).

Criação do separador,

Esta acção origina a criação de um separador na lista de separadores. As estruturas obtidas representam-se na Fig 3.39:

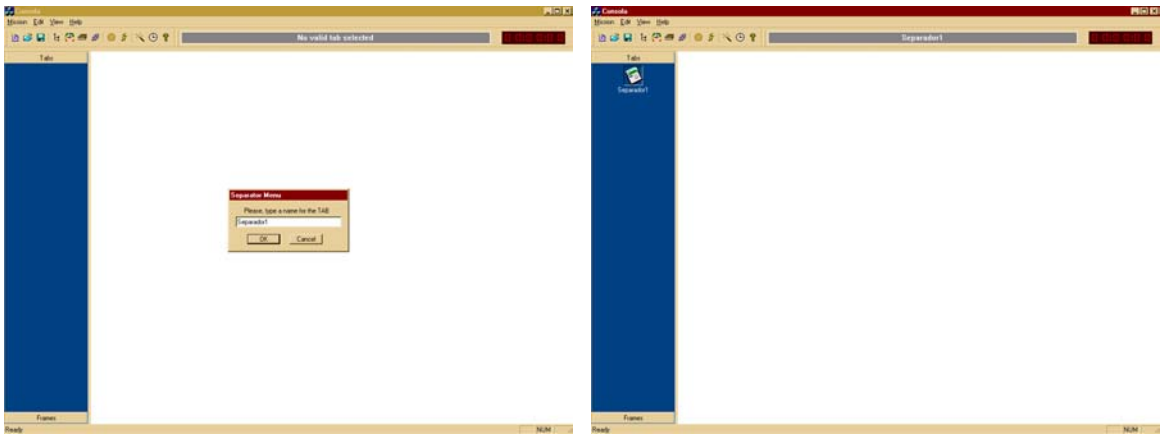

Fig. 3.38 – Criação de separador (Interface)

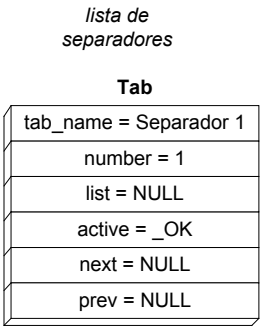

Fig. 3.39 – Criação de separador (Estruturδa de dados)

No *Separador1* vamos definir um manómetro angular com 3 escalas (as escalas são identificadas de 0 a 2), a que chamaremos manómetro1.

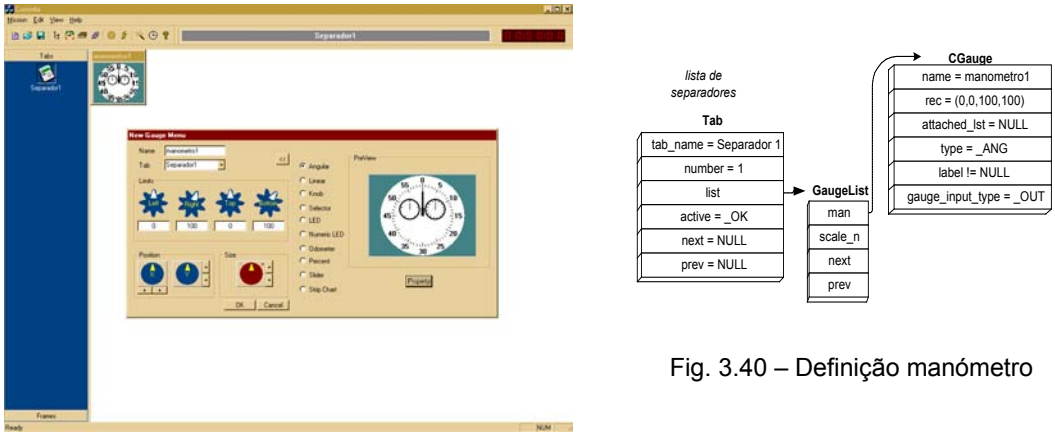

Fig. 3.40 – Definição manómetro

De seguida é definida uma trama com 4 campos. Como se pode observar na Fig.3.41, o campo *time* está sempre definido por defeito. A trama quando é criada, é actualizada nas duas tabelas de hash. O tamanho total da trama (em bytes) é calculado.

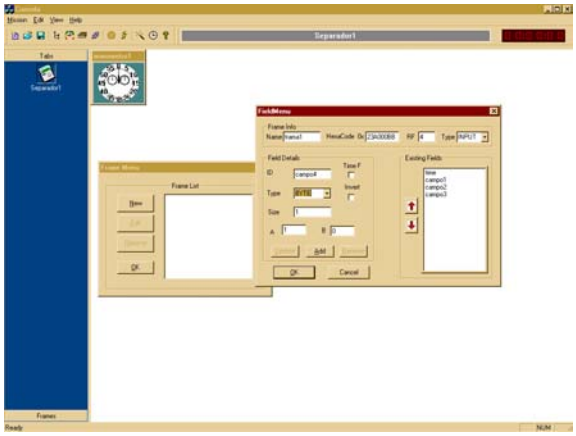

Fig. 3.41 – Criação de trama

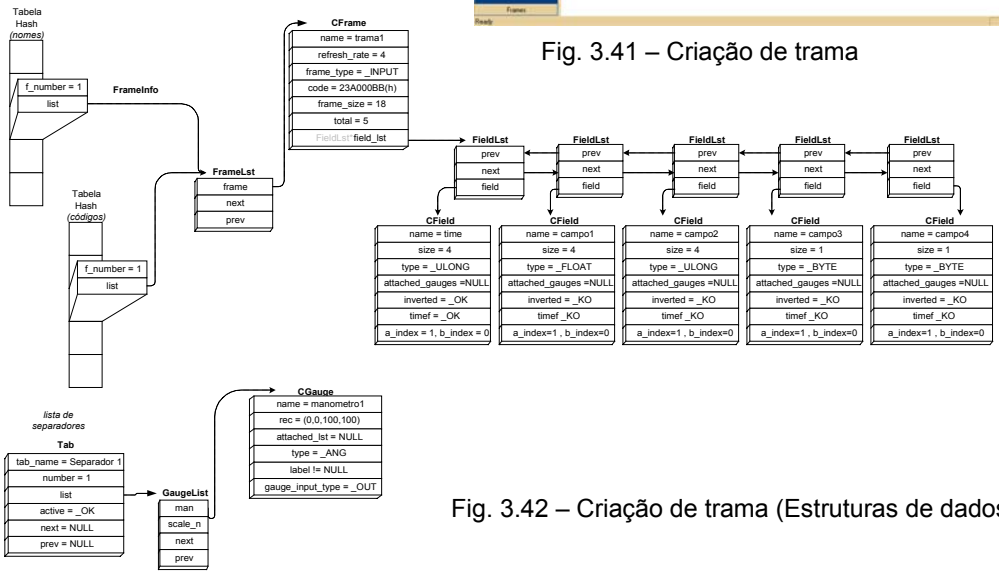

Fig. 3.42 – Criação de trama (Estruturas de dados)

Por último, vamos associar o campo 2 da trama às escalas 0 e 1 do manómetro e o campo 4 à escala 2 do manómetro.

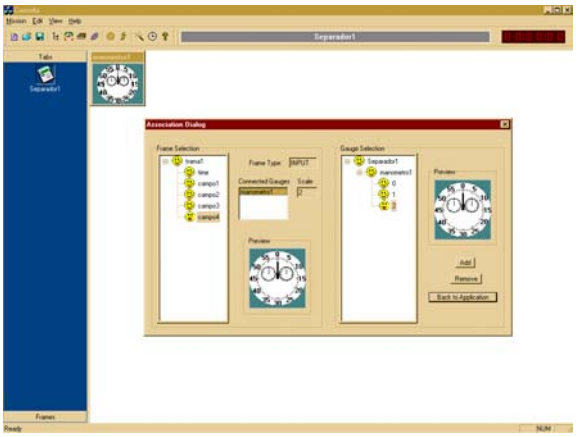

Fig. 3.43 – Associação trama/manómetro (Interface)

Como se pode observar pela Fig. 3.43, o manómetro possui 3 escalas (de 0 a 2). Confirmando as associações, as estruturas de dados actualizam-se segundo a Fig. 3.44.

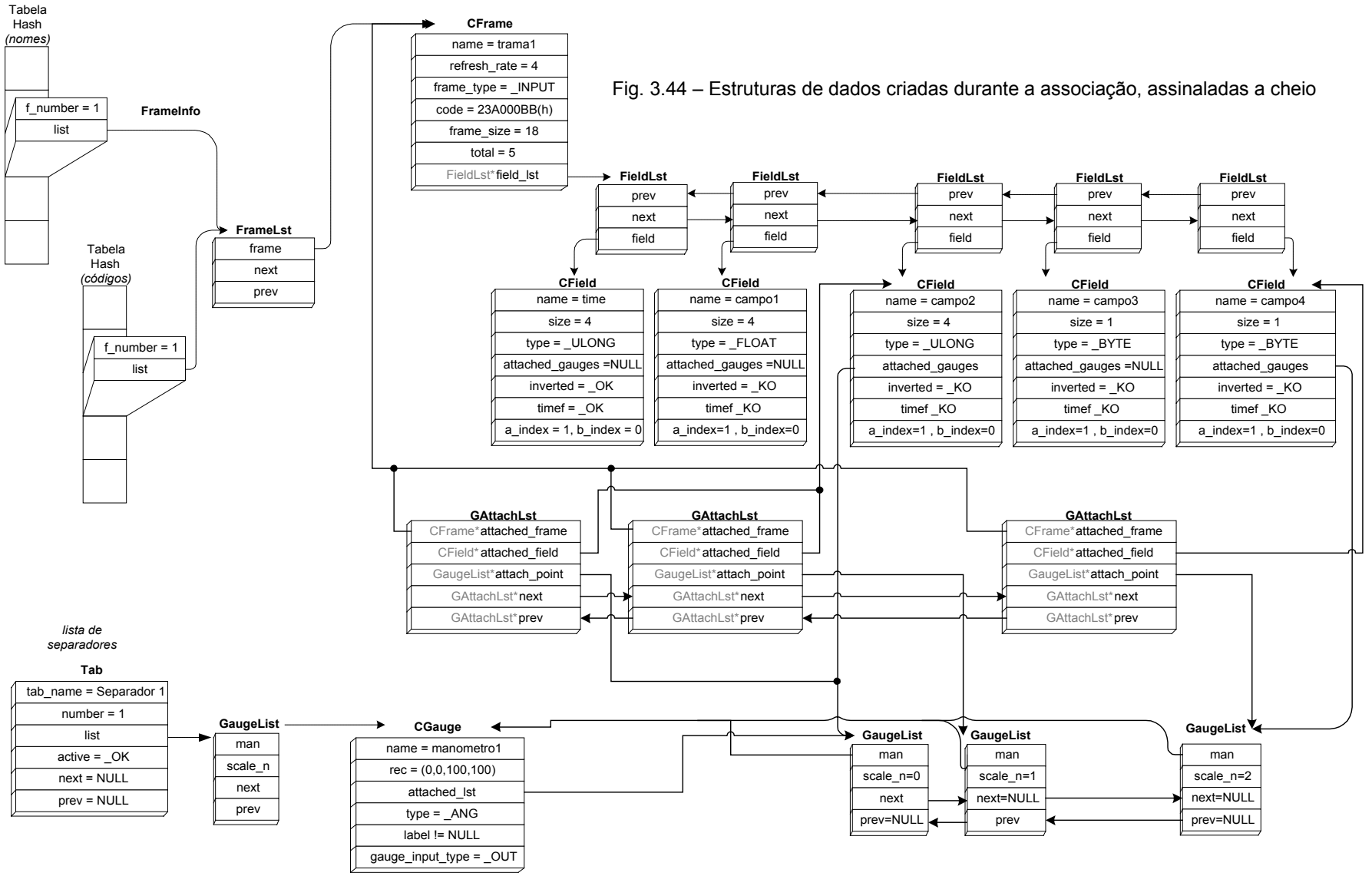

Fig. 3.44 – Estruturas de dados criadas durante a associação, assinaladas a cheio

3.4 Comunicações

As comunicações entre a consola e o veículo são feitas recorrendo ao uso de sockets do tipo datagrama. Esta foi uma especificação imposta, de forma a podermos testar o nosso trabalho no âmbito do projecto ASIMOV. É usado um socket para leitura de dados e outro para escrita, ou seja, um socket onde estão continuamente a chegar tramas vindas do veículo e outro para enviar as tramas de comando. Quando as tramas chegam ao socket, a aplicação é notificada, pelo que inicia o tratamento de informação que consiste nos seguintes passos:

1. leitura do conteúdo do socket para um buffer a partir do qual toda a informação vai ser trabalhada;
2. leitura de um campo de tamanho fixo (4 bytes) constituinte do código da trama;
3. pesquisa na tabela e HASH do código com vista a verificar se existe alguma trama definida com aquele código;
4. em caso afirmativo, o processo requer um *lock* para entrar em zona crítica de execução, após o que escreve o buffer num ficheiro de logging e procede à leitura dos campos da rama e respectivo refrescamento;
5. por último liberta o *lock* da zona crítica.

Todo este tratamento é feito por uma *thread*, automaticamente lançada quando o socket é notificado com a chegada de nova trama. Deste modo, a aplicação principal continua disponível ao mesmo tempo que a referida *thread* se encarrega de refrescar os manómetros existentes. Para o processo de envio, a trama é formada quando o utilizador interage com um manómetro do tipo INPUT, pelo que em caso de estar associado a um campo de trama, todos os campos irão ser lidos através dos manómetros associados (em caso de não estarem associados o valor por defeito é 0). Este tratamento é feito pela aplicação principal, pois é o utilizador que desencadeia esta acção.

3.4.1 Mecanismo de recepção e descodificação da trama

A principal dificuldade na implementação da descodificação da trama, consiste no fato de não se conhecer o formato das tramas. Estas são definidas pelo utilizador em *run-time*, pelo que o seu tamanho e estrutura não são conhecidas pela aplicação. Daí a necessidade de ler a informação campo a campo. Após a descodificação e pesquisa na tabela de HASH de códigos, caso a trama esteja definida, a aplicação fica com uma referência para a estrutura da trama onde pode obter

informação sobre o seu tamanho total (em bytes) e sobre a lista de campos que a compõem. Assim, irá percorrer essa lista de campos um a um, lendo os valores para a cada um deles e refrescando os manómetro associados. Cada campo é constituído por um tipo base, pelo que o número de bytes a ler do buffer é conhecido.

Após a leitura da informação do campo é aplicado um tratamento, consoante as *flags* definidas para o campo. A flag *inverted* implica a inversão da estrutura lida devido a aspectos de compatibilidade entre o hardware. Os campos *a* e *b* representam os índices de uma função do tipo *ax+b* a aplicar ao valor lido, antes de refrescar os manómetros. A flag *timef* representa uma descodificação a fazer para aplicar ao campo time de modo a transformar o número de segundos num formato HH:MM:SS.

Percebe-se que a escolha de usar uma tabela de HASH para códigos prende-se com o objectivo de diminuir o tempo de procura da trama com vista a acelerar o tratamento da informação, pois esta fase tem de ser rápida para que o utilizador esteja a ver os dados a fluir de forma coerente e eficaz.

3.4.1.1 Sincronização

É necessário que a aplicação garanta consistência na visualização de dados. Para tal recorre-se ao uso de *locks,* com vista a implementar a sincronização pretendida. Consideramos existir uma zona crítica da aplicação, desde o momento em que irá ser feito o logging e é necessário aceder ao disco, até à fase em que os manómetros são refrescados. É para entrar nesta zona crítica que o processo de tratamento ficará suspenso até ter autorização. Apenas um processo poderá executar a zona crítica, após a qual liberta o recurso e termina. Os processos, enquanto suspensos, ficam armazenados numa fila do tipo FIFO consoante a ordem de chegada, o que garante a ordem de execução correcta.

3.4.1.2 Logging

O logging é feito da seguinte forma: Sempre que o utilizador inicia a missão, é criado um ficheiro de texto cujo nome consiste em "OUT*dia_hora_min*.log*"*. Este ficheiro irá conter a estrutura binária das tramas, separadas pela string "*ASIMOVISRIST"*. Para cada 10000 tramas é criado um novo ficheiro. Um dos motivos necessários à sincronização é precisamente o facto da escrita em disco da informação, de modo a evitar acesso sobrepostos o que levaria à perda de informação.

3.4.2 Criação da Trama de Envio

Quando o utilizador altera o estado de um manómetro do tipo *INPUT*, a aplicação verifica se o manómetro esta associado. Em caso afirmativo, na estrutura da associação existe um ponteiro para a trama respectiva, pelo que todos os campos da trama irão ser percorridos. É nesta fase que se verificam se esses campos estão associados a algum manómetro, com vista a ler o seu valor. Caso estejam, o valor será lido e actualizado num buffer que se vai preenchendo à medida que se percorrem os campos. Caso contrário, o buffer é à mesma actualizado com o valor 0. Desta forma garante-se a criação, pela ordem correcta, de um buffer pronto a enviar à trama. No final acrescenta-se o cabeçalho com o código e selo temporal e envia-se através do socket de envio para um IP:PORTO destino. Este IP's são configuráveis no menu *Setup*.

4 RESULTADOS

Tal como já referimos, pretendeu-se que o sistema a desenvolver fosse um sistema genérico, rápido, robusto, flexível, simples, amigável e que disponibilizasse todas as operações inicialmente especificadas. Um outro ponto extremamente importante foi o de terem sido cumpridos os prazos inicialmente estipulados e que passavam pela conclusão do processo de desenvolvimento da aplicação antes da saída do grupo de robótica submarina do ISR para a ilha Terceira, Açores, onde seriam efectuados os testes de mar do catamaran *Delfim*, de forma a podermos testar o nosso trabalho num ambiente real.

Para tal foi definida uma metodologia de trabalho que foi seguida à risca de modo a podermos atingir todos estes objectivos. No que diz respeito à qualidade da própria aplicação foi dispendido, durante o processo de desenvolvimento, um esforço na tentativa de encontrar soluções ricas e eficientes.

Na fase de testes foi ainda desenvolvida uma aplicação que permitiu testar o nosso trabalho utilizando dados referentes a missões já realizadas anteriormente pelo catamaran do ISR. Esta aplicação, de nome *Replay*, simulava o processo de envio de tramas de dados do veículo (ver Fig. 4.1). Possibilitava alterar a velocidade de envio de tramas com vista a aproximar-se do

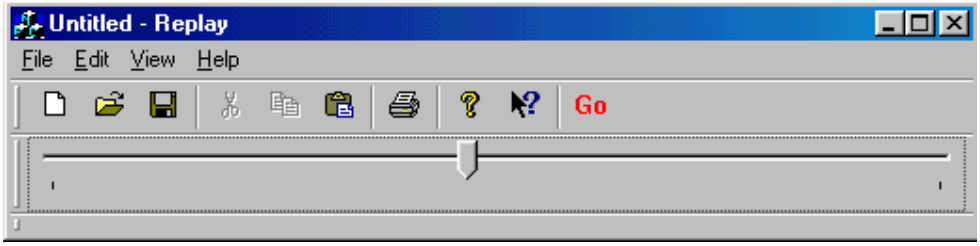

Fig. 4.1 – Front-end da aplicação Replay

ambiente real de testes. Para tal, disponibilizava uma *slide bar*.

A utilização desta aplicação permitiu-nos detectar inumeros problemas antes da saída para os testes de mar no banco D. João de Castro. Os testes seguintes foram feitos em laboratório, usando os dados enviados pelos sensores do veículo o que nos permitiu uma boa aproximação ao teste final.

A única referência em termos de resultados concretos que podemos referir, é o teste final, realizado no banco D. João de Castro nos Açores. Consideramos que o objectivo para o qual nos propusemos foi atingido, assim como todos os prazos definidos ao longo do projecto. De seguida mostramos algumas figuras da última missão do Delfim.

Fig. 4.2 – Açores, Agosto 1999

5 CONCLUSÕES

Este documento pretendeu descrever com generalidade o desenvolvimento do Trabalho Final de Curso devotado ao desenvolvimento de um Sistema Genérico de Geração de Consolas para Robótica Submarina, o qual nos permitiu aplicar conhecimentos adquiridos ao longo do curso, mas que até então haviam sido abordados apenas de forma superficial ou no plano teórico.

Na fase de definição foi documentado o conceito, os requisitos e os modelos base que serviram de suporte ao desenvolvimento desta aplicação. Com o tempo e à medida que se desenrolaram os vários ciclos de desenvolvimento houve algumas alterações à especificação inicialmente apresentada, com vista a melhorar o resultado final.

Na fase de desenvolvimento foi concebido um interpretador de script que posteriormente deu lugar a um sistema de configuração baseadas numa interface gráfica integrada na aplicação. Na fase de entrega preparou-se o produto para a apresentação, finalizando-se a documentação e o sistema.

Fazendo uma análise global ao funcionamento do sistema implementado, concluímos que os objectivos inicialmente propostos foram atingidos. Além destes objectivos, foi criada ainda uma aplicação que permitiu o teste do sistema em laboratório e a reprodução de missões previamente realizadas.

De um modo geral, sentimo-nos bastante satisfeitos com os resultados deste trabalho. A experiência foi extremamente positiva. Tivemos a oportunidade de, pela primeira vez, nos envolvermos num projecto cujos resultados podem vir a ser úteis – neste caso, para a Robótica Oceanográfica.

Referimos ainda que a oportunidade que nos foi dada pelo Instituto de Sistemas e Robótica em poder testar o nosso trabalho com o veículo *Delfim* constituiu para nós uma excelente recompensa por todo o esforço dispendido durante o último ano.

6. REFERÊNCIAS

6.1 Bibliografia

[1] Roger S., Software Engineering: a Practitioner's Approach, Pressman, 1996, McGraw-Hill (ISE Editions). ISBN: 0-0711-4603-2.

[2] Mayhew, Deborah J., Principles and guidelines in software user interface design, Prentice-Hall, 1992, ISBN: 0-13-721929-6.

[3] Shneiderman, Ben, Designing the User Interface – Strategies for Effective Human Computer Interaction, 3rd ed, Addison Wesley Longman, 1998. ISBN: 0-201-69497-2

[4] Preece, Jenny et al, Human – Computer Interaction, Addison Wesley Longman, 1994. ISBN: 0-201-62769-8.

[5] B. Kernighan e D. Richie. The C Programming Language, 2nd Ed.,Prentice-Hall.

[6] N. Wirth. Algorithms and Data Structures. Prentice-Hall, 1986.

[7] L. L. Peterson and B. S. Davie, Computer Networks, Morgan Kaufmann Publishers, 2000. ISBN 1-55860-514-2

[8] Microsoft Visual C++ 6.0, MFC Library Reference Part 1, Microsoft Press, ISBN 1-57231-865-1

[9] Microsoft Visual C++ 6.0, MFC Library Reference Part 2, Microsoft Press, ISBN 1-57231-865-7

[10] Microsoft Visual C++ 6.0, MFC Programmers Guide, Microsoft Press

6.2 Web sites

[11] CodeGuru Homepage http://www.codeguru.com

[12] University of South California: - Center for Software Engineering http://sunset.usc.edu

[13] Microsoft, Corp http://www.microsoft.com
http://msdn.microsoft.com

[14] GlobalMagicSoftware http://www.globalmagic.com

[15] XRosyGUI ™ http://www.hexatech.com

[16] Borland, Inc http://www.borland.com

[17] ComponentGarden Website http://www.componentgarden.com

[18] Bumble Bee Software http://www.bumblebeesoftware.com

7 APÊNDICES