Aplicação de Métodos Estruturados na Especificação e Implementação de um Sistema de Comunicação em Tempo Real

José P. Santos, Fernando M. S. Ramos

Resumo- **Em 1990 a Telecom Portugal contratou o INESC (Instituto de Engenharia de Sistemas e Computadores) para desenvolver o protótipo de um Sistema de Televigilância suportado pela RDIS (Rede Digital com Integração de Serviços) no âmbito do apoio às actividades de investigação e desenvolvimento promovidas pela Telecom Portugal relacionadas com a introdução em Portugal da RDIS, projecto que foi concluído em 1994. Este artigo inclui uma breve descrição e discussão sobre a aplicação de técnicas estruturadas de desenvolvimento e especificação de sistemas de informação ao sistema implementado, sendo apresentada a análise e o projecto estruturado do software para processamento em tempo real desenvolvido para o componente mais complexo do sistema a Estação Central de Televigilância.**

Abstract- **In 1990 INESC (Institute of System Engineering and Computers) was contracted by PT-Portugal Telecom - the Portuguese telecom operator - to develop the prototype of a telesurveilling system supported on ISDN (Integrated Services Digital Network), in the context of the R&D activities promoted by PT concerning the introduction of ISDN in Portugal, project that was concluded in 1994. This paper includes a brief survey and discussion on the structured techniques evaluated for the system specification, and presents and discusses the analysis and design processes of the software developed to support the real time functionalities of the system (images alarms,...) focused on the most complex of its components, the Central Telesurveilling Station**

I. DESCRIÇÃO DO SISTEMA IMPLEMENTADO

O Sistema de Televigilância [1] tem como objectivo permitir a visualização numa estação central de imagens obtidas por câmaras de vídeo localizadas em pontos remotos.

A arquitectura do sistema desenvolvido baseia-se em dois tipos de estações: Estação Central (EC) e Estação Remota (ER); ambas as estações baseiam-se em plataformas PC compatível com placas de comunicação RDIS (2B+D) e placas de processamento de imagem.

A comunicação entre a EC e a(s) ER(s) baseia-se no canal B (64Kbps), no modo de comutação de circuitos, para transferir imagens e mensagens de controlo do sistema, enquanto no canal D (16Kbps) são transferidas mensagens sobre os alarmes usando a capacidade UUS - User to User Signalling disponibilizada pela RDIS.

Este sistema permite transferir dois tipos de imagens entre a ER e a EC: imagens em movimento a preto e branco com 144x180 pixel com uma taxa de transferência de oito imagens por segundo, e imagens a cores com 288x360 pixel com uma taxa de transferência máxima de 1 imagem por segundo.

O estabelecimento de ligação entre uma ER e a EC pode ser da iniciativa de qualquer uma delas: a iniciativa é da ER quando um dos seus sensores (intrusão, fogo,...) fôr accionado, e é da EC quando o seu utilizador pretender configurar ou obter imagens de uma ER.

Mesmo quando a ligação entre a EC e uma ER está estabelecida a EC tem a capacidade de receber e processar indicações de alarmes que ocorram nessa ou noutra ER: nesse caso o utilizador é informado dessa(s) ocorrência(s) podendo optar por atender mais tarde ou atender imediatamente esse alarme passando nessa altura a obter imagens dessa ER.

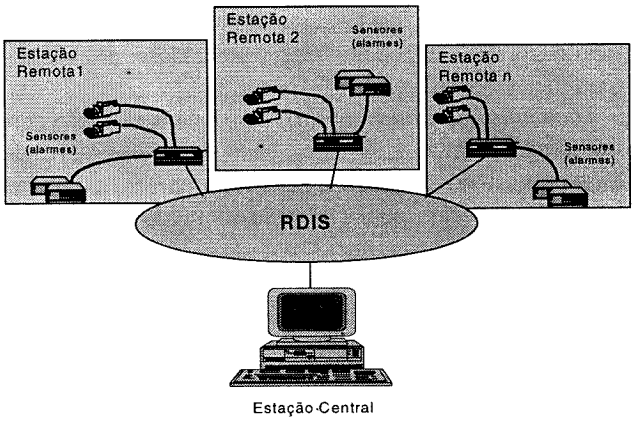

Fig. 1 - Sistema de Televigilância

A partir das funcionalidades do sistema torna-se claro que diversos tipos de ocorrências devem ser tratadas em tempo real pela EC, nomeadamente comunicações bidireccionais (envio de comandos para a(s) ER(s)e recepção de confirmações, alarmes e imagens), processamento dos alarmes, reconstrução de imagens e visualização, bem como interacções com o utilizador. Pelas mesmas razões a ER tem de ser capaz de processar diversas ocorrências em tempo real, nomeadamente comunicações bidireccionais (transmissão de alarmes e

imagens e recepção de comandos) e aquisição e compressão de imagens.

Este artigo descreve e discute o processo de análise, projecto estruturado e implementação da EC.

II. MÉTODOS ESTRUTURADOS PARA DESENVOLVIMENTO DE SOFTWARE

O uso de métodos formais de análise e projecto estruturado aplicados ao desenvolvimento de software é necessário para permitir um desenvolvimento estruturado e uma descrição rigorosa e completa do software a desenvolver.

A análise do sistema inclui a definição do seu interface com o exterior (nomeadamente a lista de todas as ocorrências provenientes do exterior que o sistema tem de tratar incluindo o formato dos dados trocados), os dados internos e dados armazenados no sistema e ainda as operações a efectuar sobre esses dados.

Para a especificação deste sistema foi feito um estudo comparativo de três métodos de análise, com o objectivo de seleccionar aquele que melhor se coadunava com o sistema a desenvolver: DeMarco [2], Gane-Sarson [3] and Yourdon [4].

DeMarco e Gane-Sarson propõem que a primeira etapa da análise de um novo sistema a desenvolver seja a criação do seu Modelo Lógico, isto é, a descrição dos dados e operações a efectuar, recorrendo aos:

Diagrama de Fluxo de Dados (DFD): ferramenta gráfica usada para mostrar os fluxos de dados, armazéns e operações a efectuar sobre os dados (processos); estes diagramas são construidos usando quatro tipos de entidades: fluxos de dados, processos, armazéns, e entidades externas.

Dicionário de Dados (DD): é uma lista organizada com a descrição (formato dos dados, gama de valores que podem assumir,...) dos fluxos de dados, armazéns e entidades externas usadas nos DFDs, permitindo uma melhor compreensão dos mesmos.

Especificação de Processos (EP): como o nome indica a especificação dos processos descreve as operações que cada processo deve efectuar sobre os dados que recebe; para esta especificação e consoante o seu nível de abstracção e tipo de processo podem ser usadas diferentes formalismos, desde o inglês estruturado até aos diagramas de transição de estados.

Para a definição dos armazéns DeMarco propõe o uso de diagramas de estrutura de dados enquanto Gane-Sarson propõe o uso das dependências funcionais com redução à 3FN-Terceira Forma Normal, sendo esta última preferível por permitir um maior nível de organização e consistência, mas obrigando ainda na análise a definir todos os dados necessários a cada armazém.

Na etapa seguinte tanto DeMarco com Gane-Sarson propõem o desenvolvimento do novo Modelo Físico, que inclui a definição do interface homem-máquina e a identificação de vários modelos físicos possíveis; a escolha do modelo a utilizar deverá ser feita com base no critério custo*benefício e avaliando o esforço requerido para o seu desenvolvimento e implementação.

Os métodos propostos por DeMarco e Gane-Sarson são habitualmente referidos como Métodos Clássicos de Análise Estruturada. Estes métodos consomem muito tempo, sobretudo quando o sistema em análise é uma evolução de um sistema já existente, dado que isso obriga também à compreensão e documentação do sistema existente. Por outro lado, estes métodos não incluem uma forma prática de descrição de sistemas em tempo real, e não possuem ferramentas que permitam, com um grau de abstracção adequado, a definição estruturada dos armazéns.

Nos anos 80 autores como McMenamim & Palmer [5], Ward & Meller [6] e em especial Yourdon [4], introduziram novos conceitos que ajudam a ultrapassar as desvantagens atrás enunciadas dos Métodos Clássicos de Análise Estruturada. Nesta nova abordagem, proposta por Yourdon e conhecida como Análise Estruturada Moderna, o processo de análise de um novo sistema começa pela definição do Modelo Essencial, o qual é constituído pelo Modelo Ambiente e pelo Modelo de Comportamento.

No Modelo Ambiente é definido o interface entre o sistema a desenvolver e o universo envolvente, e ainda a lista de acontecimentos, incluindo os dados associados, que o sistema tem de tratar.

No Modelo de Comportamento é descrito o comportamento interno do sistema com base na lista de acontecimentos, sendo desenvolvido um DFD de nível intermédio para cada um desses acontecimentos (*event partitioning*), ao contrário da abordagem top-down usada pelos outros métodos. As ferramentas utilizadas são os DFDs, os DDs, as EPs e uma nova ferramenta:

Diagrama de Entidade-Relacionamento (DER): diagrama que permite a criação de armazéns normalizados e consistentes com base nas entidades do sistema e no relacionamento entre os seus dados associados.

Por outro lado Yourdon propõe para a especificação dos processos o uso de Diagramas de Transição de Estados e das *Pre-post Conditions*:

Diagramas de Transição de Estados (DTE): diagramas que identificam os diferentes estados em que (neste contexto) um processo se pode encontrar e com base nesse estado as operações a exercer sobre os fluxos de dados. A estes processos chamam-se processos de controlo.

Pre-post Conditions (PPC): definem, com um grande nível de abstracção, as operações a efectuar sobre os fluxos de dados por um processo.

Os DTEs fornecem à análise uma ferramenta poderosa para a descrição dos processos de controlo que são essenciais para a especificação de sistemas de tempo real. Os DER permitem a definição dos armazéns com um

elevado nível de abstracção e organização sendo por isso uma importante ferramenta da análise.

No Modelo Ambiente para permitir uma separação clara entre o ambiente envolvente e o sistema em análise é definido o diagrama de contexto, que permite definir claramente qual a parte (o todo ou uma parte) do sistema vai ser objecto de especificação.

No desenvolvimento do modelo de comportamento Yourdon propõe a abordagem *Event Partioning* já referida, o que facilita a análise do sistema dado que orienta a especificação com base nos acontecimentos.

Para a passagem da análise para o projecto estrututrado Yourdon propõe a criação dos seguintes modelos: O Modelo de Implementação do Sistema, que inclui o Modelo do Processador e o Modelo da Tarefa, e ainda o Modelo de Implementação do Programa.

O Modelo do Processador consiste na definição da melhor estratégia de alocação das especificações funcionais criadas através da análise pelos vários processadores/máquinas disponíveis para a implementação do sistema. Por sua vez o Modelo da Tarefa define para cada processador/máquina a(s) tarefa(s) a desempenhar. Cada tarefa fica então com um conjunto de DFDs, armazéns, etc. alocados, sendo então necessário converter esse modelo assíncrono num modelo síncrono, dado a implementação de uma tarefa (em UNIX ou DOS) ser efectuada de uma forma síncrona. Com este objectivo Yourdon propõe a criação do Modelo de Implementação do Programa usando para isso uma ferramenta gráfica conhecida por Diagrama de Estrutura [7].

III. ESTUDO DE UM CASO: APLICAÇÃO DE MÉTODOS ESTRUTURADOS PARA O DESENVOLVIMENTO DE UM SISTEMA DE TELEVIGILÂNCIA SUPORTADO PELA RDIS

A figura 2 apresenta o Modelo Ambiente do sistema em análise, cuja descrição funcional foi resumidamente apresentada na secção I:

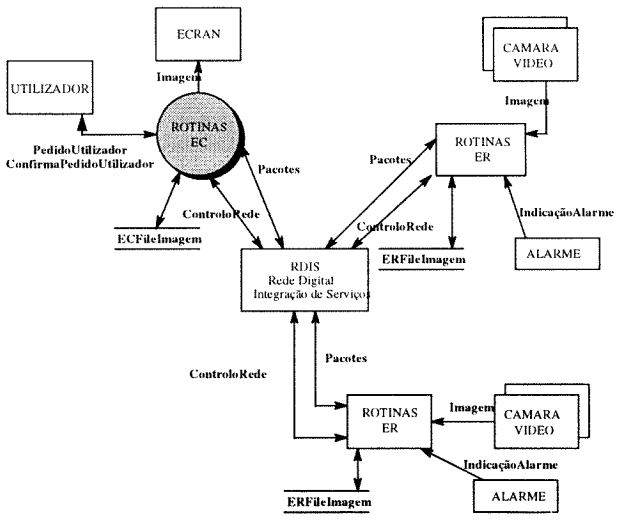

Fig. 2 - Modelo Ambiente do Sistema de Televigilância suportado pela RDIS.

As imagens obtidas a partir de câmaras de vídeo são adquiridas e comprimidas através de placas de aquisição e compressão de imagem em cada ER, após o que são transmitidas para a EC através da placa RDIS, o que envolve o envio de um ou mais pacotes de dados. Se a ligação não se encontrar ainda estabelecida entre a ER e a EC, a ER guarda localmente imagens, para uma eventual visualização posterior, no armazém ERFileImagem.

Na EC as imagens são recebidas da rede através da placa RDIS, e visualizadas após terem sido descomprimidas. O utilizador pode adicionalmente pedir para gravar as imagens na EC, o que será feito no armazém ECFileImagem.

A EC e as ERs podem também trocar outro tipo de informação, tal como comandos de configuração e indicação de alarmes.

Neste artigo apenas é abordada a análise e o projecto da EC, o que está expresso no Diagrama de Contexto apresentado na figura 3.

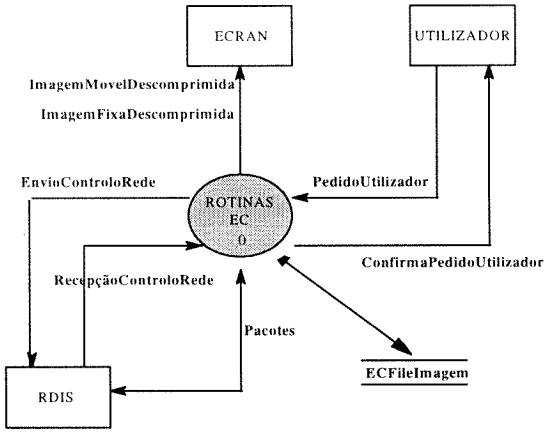

Fig. 3 - Diagrama de Contexto da EC.

Os fluxos de dados recebidos e as correspondentes respostas são descritos no DD. As mensagens recebidas provenientes do utilizador ou da ER não são mais que os acontecimentos que o sistema tem de tratar.

Os acontecimentos gerados pelo utilizador podem ser de quatro tipos diferentes: pedido de configuração da EC, pedido de configuração de uma ER, operação de uma ER, e visualização de imagens na EC. Os acontecimentos recebidos da rede vindos da ER podem ser de dois tipos: indicação de alarme numa ER e recepção de imagens.

Do Modelo Ambiente também faz parte a lista de todos os acontecimentos que o sistema tem de tratar.

O Modelo de Comportamento descreve o comportamento interno do sistema em análise baseando-se na lista de acontecimentos; para cada acontecimento é criado um DFD de nível intermédio (*Event Partitioning*) para permitir definir a forma como cada acontecimento (fluxo de dados) deve ser processado e/ou armazenado. Com este objectivo são utilizadas ferramentas como os DFD, DD e EP. A figura 4 apresenta o DFD de mais alto nível correspondente ao Diagrama de Contexto, que foi contruído com base nos DFDs de nível intermédio.

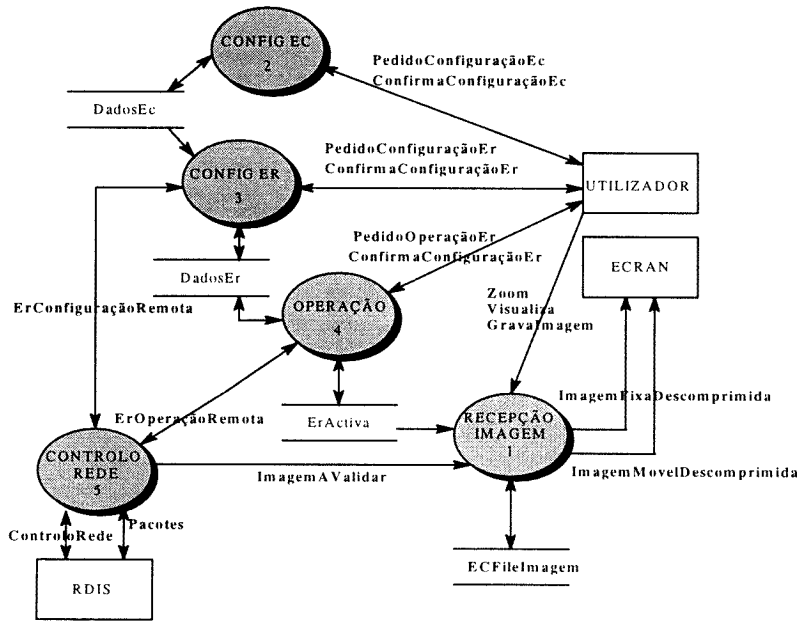

Fig. 4 - DFD 0 da EC.

A figura 5 apresenta exemplos de entradas no DD relacionadas com o DFD0. O DD inclui entradas semelhantes para todas as estruturas de dados existentes no sistema.

```
ImagemAValidar: receção de uma imagem vinda da ER.
ImagemAValidar= 5555
        + NumeroCâmara              = {byte}1  = 0x01-0x08
        + NumeroFragImagem          = {byte}1  = 0x00-0xff
        + Imagem                    =1{byte}26000
```

Fig. 5 - Um exemplo de uma entrada do DD.

As técnicas usadas na especificação dos processos incluem as *Pré-post Conditions*, Inglês estruturado e Diagramas de Transição de Estados.

A figura 6 mostra como exemplo a especificação do processo 1 - *Recepção Imagem* apresentado no DFD0 (figura 4).

```
BEGIN
IF GravaImagemFixa
        Copia a imagem do ficheiro (default.jpg) para a ficheiro
        indicado
ELSE
BEGIN
    IFImagemAValidar WITH NúmeroCâmara=NumeroCâmaraActiva
        IF NumeroCâmaraActiva= 1 or 2
            Transfer imagem fixa comprimida para a placa de imagem
            Guarda imagem fixa comprimida no ficheiro (default.jpg)
        ELSE (* 2 < NumeroCâmaraActiva < 9 *)
            Transfer imagem móvel comprimida para a placa de
            imagem
        ENDIF
    ENDIF

    IF Show
        Abre file indicada
        Transfer imagem fixa comprimida para a placa de imagem
    ENDIF

    Inicia a descompressão da imagem
    DO WHILE (descompressão em progresso)
    ENDDO
    Transfere imagem da placa de imagem para a memoria
    Visualiza imagem
END
END
```

Fig. 6 - Exemplo da especificação de um processo usando Inglês estruturado.

Depois do processo de análise ter sido concluido foi iniciado o projecto estruturado. Como era suposto a EC ser implementada numa única máquina com um único processador e com base no sistema operativo MS-DOS, não há necessidade de criar o Modelo de Implementação do Sistema, dado todas as especificações criadas na análise serem alocadas a uma única tarefa numa única máquina/processador, sendo apenas necessário criar o Modelo de Implementação do Programa. A figura 8 apresenta uma versão simplificada do Diagrama de Estrutura inicial do Modelo de Implementação do Programa já com um conjunto de refinamentos necessários para evitar problemas de recursividade indirecta e para garantir o processamento adequado dos acontecimentos assíncronos que ocorrem em sistemas de processamento em tempo real.

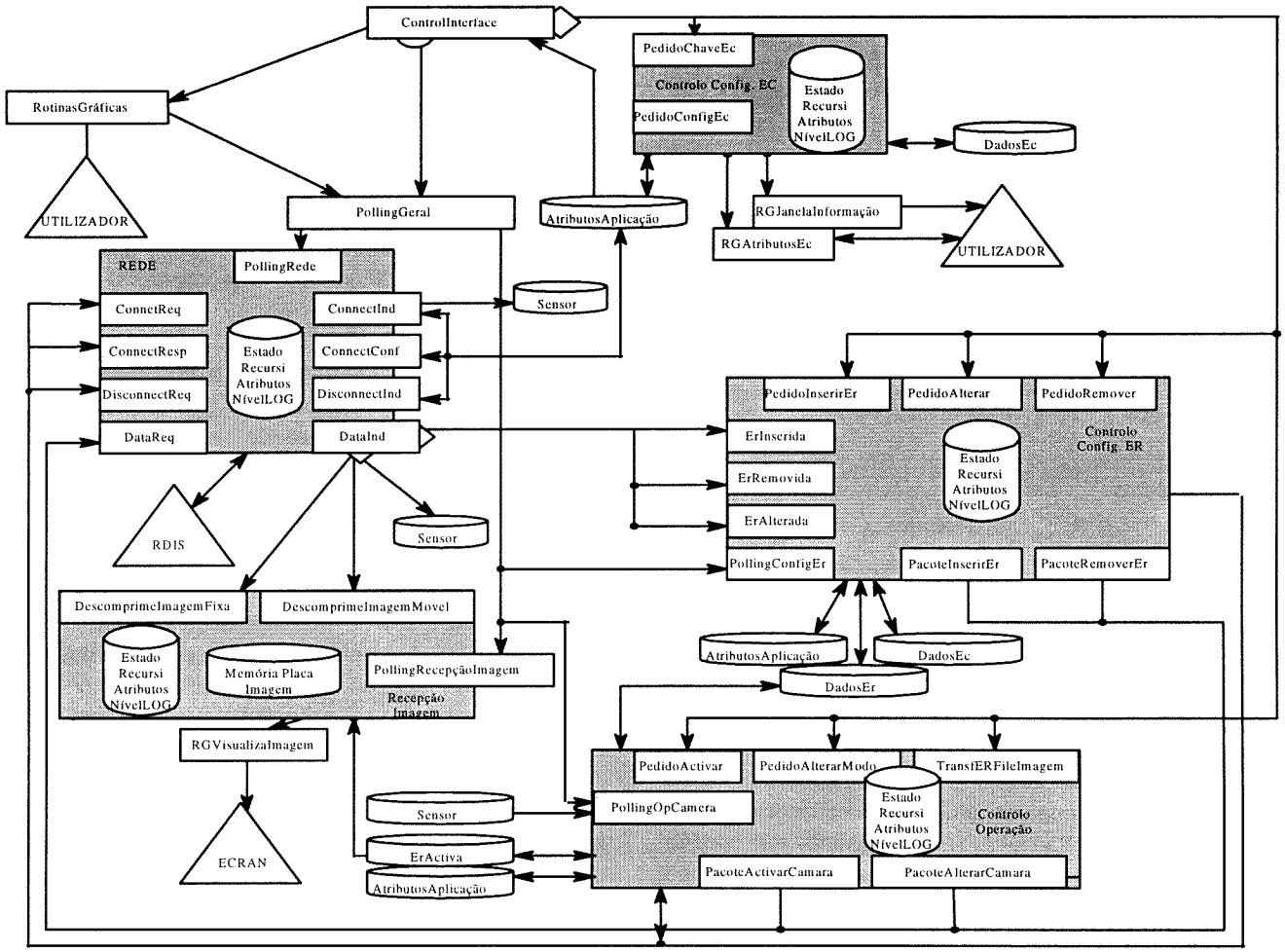

Fig. 7 -Diagrama de Estrutura inicial do Modelo de Implementação do Programa da EC.

Nesta figura surgem novos módulos, sub-módulos e armazéns adicionais que não foram definidos no Diagrama de Contexto: este facto deve-se à necessidade de construir um modelo síncrono necessário à implementação, com base no modelo assíncrono da análise.

Assim os diagramas das figuras 7 e 8 já refletem os seguintes aspectos:

1 -A separação entre os três tipos base de rotinas, rotinas de *Interface*, rotinas de *Comunicação* e rotinas de *Processamento* (estas últimas relativas aos três módulos de controlo e ao módulo de recepção de imagem), para permitir um desenvolvimento em paralelo da aplicação por uma equipa de programadores, facilitando a integração e os testes finais, uma vez que é possível realizar testes individuais. A própria integração é gradual uma vez que cada módulo quando atinge alguma maturidade é passado aos outros programadores. Por outro a separação entre os três facilita a reutilização de módulos.

2- Para se obter uma melhor organização do software, cada tipo de rotinas deve ser implementada no seu directório respectivo. Cada um destes módulos, ao ser implementado, origina diversas rotinas de alto nível e um conjunto de rotinas auxiliares, eventualmente de processamento intermédio. As rotinas de mais alto nível que servem de interface com o exterior devem ser todas agrupadas num único ficheiro de interface, para que esse módulo possa ser usado de uma forma clara e simples pelos restantes, sendo as rotinas auxiliares agrupadas noutros ficheiros nesse mesmo directório. Assim cada módulo principal deve oferecer para o exterior um conjunto específico de rotinas (ou métodos se o módulo for implementados à custa de um objecto) onde cada rotina corresponda a um pedido a processar e onde os seus parâmetros sejam, de preferência, parâmetros elementares em vez de ponteiros para estruturas, dado que isso implicaria o conhecimento prévio dessas estruturas por parte da aplicação cliente. Este aspecto de organização está refletido nas zonas a sombreado.

3- A relação entre as rotinas e as variáveis que estas manipulam, deve ser clara para quem examinar o software desenvolvido.

4- Como algumas acções levam tempo a serem concluídas e a execução do programa não deve ficar suspensa durante esse período (porque podem existir outros pedidos à espera de serem processados), é

necessário que cada módulo, implementado ou não à custa de máquinas de estado eventualmente só com um estado (fig. 8), possua um conjunto de variáveis (atributos) que lhe permita continuar a processar novos pedidos não deixando, no entanto, de possuir informação suficiente para processar os anteriores logo que possível.

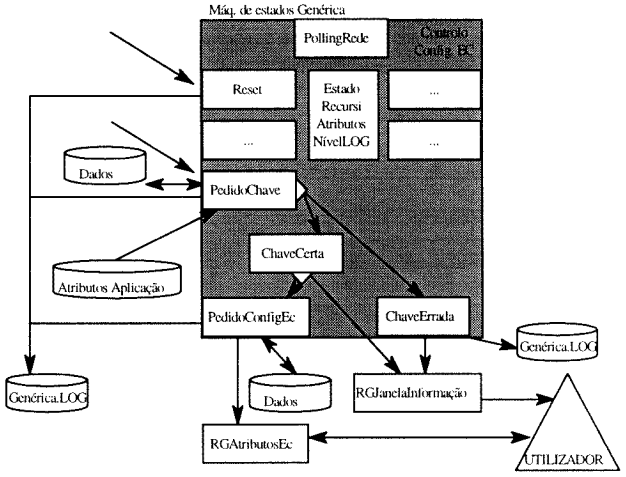

Fig. 8 - Exemplo da arquitectura de uma máquina de estados Genérica.

Para isso o módulo deve ainda possuir uma rotina (ex. *PollingRede*) que possa ser chamada periodicamente pela aplicação para que, quando existirem pedidos que ela não possa processar de uma forma total ou imediata por não existirem recursos disponíveis naquela altura, ou a máquina de estados não se encontrar no estado adequado para o fazer e necessitar de colocar esse pedido numa fila de espera, o pedido não deixe de ser processado. Nessa altura é necessário que o módulo possua um contador interno para que possa esperar algum tempo (< *TimeOut*) antes de tentar processar de novo aquele pedido ou simplesmente completar o seu processamento.

Cada rotina (uma rotina para cada pedido possível) deste módulo deve retornar um de três valores possíveis: o pedido não pode ser aceite, o pedido foi processado e o pedido irá ser processado. Neste último caso o pedido irá ser processado se a aplicação evocar periodicamente a rotina de *polling* da máquina de estados ou se a aplicação evocar outros procedimentos que venham a despoletar outra rotina desta máquina de estados que complete o seu processamento. A figura 8 mostra um exemplo de implementação de um módulo genérico.

5- O processamento de acontecimentos em sistemas de tempo real deve prevêr a chegada de novos acontecimentos para processar ainda antes do anterior ter sido completamente processado, nomeadamente quando existem pedidos cujo o processamento é demorado. No caso do Sistema de Televigilância, que é objecto deste trabalho, vários problemas se levantaram e a arquitectura do *software* é crítica. Os problemas que se podem colocar são de diferentes tipos, como por exemplo:

- Quando a meio do processamento de um pedido feito pelo utilizador chega da rede, um aviso de que um sensor foi accionado, ou chegam da rede imagens para descomprimir.
- Quando a EC está a descomprimir imagens vindas da rede pode chegar um aviso de que um sensor foi accionado; como este aviso não pode ser ignorado é necessário continuar a monitorar a rede e processar o alarme em paralelo com a descompressão da imagem.
- Como a rede tem de estar sempre a ser monitorada, mesmo durante a descompressão de uma imagem (fixa ou móvel) pode chegar uma nova imagem para descodificar ainda antes da anterior ter sido totalmente descodificada.

Duas hipóteses se põem na recepção dos acontecimentos: ou os dados recebidos são armazenados num *buffer* de recepção e processados posteriormente quando houver disponibilidade da aplicação, ou os dados são imediatamente processados, interrompendo-se momentaneamente o processamento anterior. Como vamos ver a seguir, ambas as hipóteses têm vantagens e desvantagens.

A primeira hipótese (processamento posterior) tem a grande desvantagem de implicar o uso de um *buffer* de recepção, com todos os atrasos, ocupação de memória e gestão de *buffers* inerentes. O tamanho do *buffer* de recepção dependerá do tempo máximo entre o processamento dos dados recebidos e da cadência de recepção de dados nesse período, podendo em casos extremos, ocorrer uma situação de *overflow*.

Apesar das desvantagens já referidas, esta hipótese tem a vantagem de evitar a recursividade indirecta que pode existir na segunda hipótese.

A segunda hipótese (processamento imediato) evita o uso do *buffer* de recepção e torna o processamento mais rápido. Nesta hipótese as rotinas de processamento não devem suspender a execução do programa enquanto esperam a conclusão do processamento mais demorado, mas devem retornar a execução ao programa chamador; neste caso deverão estar preparadas para que a aplicação as possa eventualmente reevocar, fazendo-lhe outro pedido ainda antes do anterior ter terminado (recursividade indirecta).

A recursividade indirecta pode originar situações anómalas em que o programa funciona durante algum tempo e sem razão aparente bloqueia ou provoca a reinicialização inesperada da máquina. Outra consequência pode ser a necessidade de compilar o programa com um *Stack* muito grande. Este tipo de problemas pode fazer o programador perder muito tempo a rever o código e a fazer alterações indevidas, até que o problema seja, finalmente, detectado. Este problema pode ser facilmente detectado usando uma variável estática que é inicializada a um no ínicio da rotina que corra o risco de

ser reevocada e posta a zero no fim; como uma variável estática mantém o seu valor entre reevocações funciona como uma "flag".

Esta discussão permite concluir que a segunda hipótese pode ser preferível dependendo da complexidade da aplicação e das precauções tomadas pelo programador.

6 -Cada módulo deve possuir um ficheiro de LOG onde todos os pedidos recebidos e todas as respostas às suas acções fiquem gravadas para consulta posterior. Este ficheiro é particularmente útil durante o desenvolvimento da aplicação devido à grande quantidade de erros de execução, mas também é útil em aplicação real para que seja sempre possível detectar com exactidão erros que venham a ocorrer. A máquina de estados deve ainda permitir vários níveis de LOG como por exemplo: registo de todas os pedidos e respostas a acções, registo apenas de pedidos que não puderam ser satisfeitos ou respostas negativas a acções por si tomadas, ou ainda registo dos parâmetros que compõem cada pedido.

7- Cada módulo deve ainda ser capaz de permitir à aplicação fazer o seu próprio *reset* para que seja possível numa situação anómala ou mesmo numa situação normal voltar ao ponto de partida.

A figura 7 apresenta os aspectos mais importantes da arquitectura proposta para a implementação do sistema. Nesta figura pode-se notar a existência de cinco máquinas de estados, cada uma delas respeitando a arquitectura atrás referida: *Controlo Config. EC*, *Controlo Config. ER*, *Controlo Operação*, *Rede* e *Recepção de Imagem*.

Pode-se ainda observar a existência de várias *Rotinas Gráficas* responsáveis por criar janelas de interface com o *Utilizador* e utilizadas pelos módulos *Controlo do Interface* e pelas máquinas de estados, embora a figura não o mostre, excepto para a máquina de estados *Controlo Config. EC* (ex. *RGJanelaInformação*, *RGAtributos* e *RGVisualizaImagem*), para que estes possam receber pedidos ou possam enviar para o *Utilizador* as mensagens e imagens necessárias. As *Rotinas Gráficas* são também responsáveis por evocar o módulo de *PollingGeral* durante o período de tempo em que o *Utilizador* introduz dados, para evitar que o sistema fique sem monitorar a rede ou sem poder processar pedidos eventualmente já recebidos mas ainda não completamente processados durante esse período de tempo.

O módulo *PollingGeral* é evocado não só pelas rotinas gráficas mas também, ciclicamente, pelo módulo *Controlo do Interface* quando o sistema se encontra à espera de novos pedidos para processar. Como se pode ver na figura 7 o módulo *PollingGeral* evoca todas as rotinas de *polling* do sistema, nomeadamente a rotina de *PollingRede* que permite ao sistema receber pedidos da rede, nomeadamente o pedido para processar a ocorrência de um sensor numa ER; quando essa situação ocorre, a rotina *DataInd* ou a rotina *ConnectInd* é evocada e os atributos relativos ao sensor (ex. NomeEr, número do sensor accionado,...) são guardados no armazém *Sensor*.

Posteriormente a rotina *PollingOpCamara* é evocada pelo *PollingGeral* permitindo à máquina de estados *Controlo Operação* consultar os atributos do armazém *Sensor*, analisar o estado da aplicação (armazém *Atributos Aplicação*) e os atributos da ER presentemente activa (armazém *ErActiva*) para processar então o sensor activando (rotina *PacoteActivarCamara*) a recepção de imagens da câmara associada.

Quando a máquina de estados *Recepção Imagem* é evocada pela rotina *DataInd*, é-lhe passada uma imagem fixa ou uma imagem móvel para descodificar. Como a descodificação de uma imagem é demorada, as rotinas de recepção de imagem colocam a imagem a descodificar na memória da placa de imagem e retornam o controlo da execução do programa para os restantes módulos (em última análise para o *Controlo do Interface*) e ciclicamente o módulo *PollingGeral* evoca o *PollingRecepçãoImagem* que depois de verificar se a imagem já foi descodificada a enviará para o *Écran*.

Embora a figura não o explicite, os pedidos feitos ao módulo de rede implicam internamente o *polling* da rede para confirmar se o pedido foi aceite. Por exemplo, quando se envia um pacote para a rede (através da rotina *DataReq*) esse pacote é colocado num *buffer* de envio que só pode ser considerado disponível para acolher um novo pacote quando a confirmação de que o anterior foi enviado tiver sido recebida, implicando por isso a monitorização (*PollingRede*) da rede; no entanto a monitorização da rede permite receber novos dados ou outros acontecimentos (nomeadamente um pacote de dados para ser processado) antes da chegada da confirmação esperada. Nessa situação dois tipos de implementação são possíveis: ou se processa imediatamente os dados recebidos o que pode originar recursividade indirecta, ou são colocados num *buffer* de recepção com toda a gestão e problemas de *buffer overflow* que essa opção implica. Neste caso optou-se por processar imediatamente os dados recebidos (*DataInd*) tendo o cuidado de prever a recursividade indirecta que este tipo de implementação pode gerar. Por exemplo, se a rotina *PacoteActivarCamara* da máquina de estados *Controlo Operação* evocar a rotina *DataReq* e durante a sua execução fôr recebido um pacote com a informação sobre a ocorrência de um alarme numa ER, nessa altura a máquina de estados *Controlo Operação* volta a ser evocada para processar o sensor e as rotinas *PacoteActivarCamara* e *DataReq* poderiam voltar a ser evocadas segunda vez dado que o processamento do sensor implica enviar um pacote com a ordem para adquirir imagens da câmara associada ao sensor accionado. Para evitar essa recursividade indirecta a ocorrência de alarmes não é imediatamente processada mas a sua ocorrência é guardada no armazém *Sensor* e posteriormente processada por *PollingOpCamara*.

IV CONCLUSÕES

Da aplicação de técnicas estruturadas no desenvolvimento e especificação de um Sistema de Televigilância baseado na RDIS, cujas funcionalidades foram sumariamente descritas neste artigo, várias conclusões sobre as suas vantagens e desvantagens podem ser apresentadas: vamos, no entanto, centrar-nos nas principais.

Primeira conclusão, provavelmente óbvia mas que não é de mais salientar, é que a utilização destas técnicas é fundamental no desenvolvimento e documentação de software, dado que permite definir e documentar soluções para problemas específicos; no entanto, estas técnicas têm um custo imediato que é o tempo consumido na sua aplicação, mas que é rentabilizado por permitir uma especificação completa e rigorosa.

O maior ou menor nível de detalhe das especificações criadas na análise e projecto estruturado deve ser pesado com o tempo que demoram a efectuar, sendo necessário encontrar uma solução de compromisso caso a caso. Nesta perspectiva é conveniente usar, sempre que possível, para a especificação dos processos as *Pré-Post Conditions* dado que permitem, com rapidez e com um elevado grau de abstracção, especificar os processos. Além disso as *Pre-Post Conditions* dão ao programador a liberdade de implementar esses processos da forma mais adequada ao ambiente de implementação

Um aspecto essencial das especificações criadas consiste na definição rigorosa e com um bom nível de abstracção das principais estruturas de dados conseguida com os DER tendo-se concluido ser esta a melhor ferramenta proposta para este fim. O Dicionários de Dados é particularmente importante quando o sistema se destina a ser implementado por uma equipa de trabalho.

Apesar das vantagens referidas estas técnicas possuem algumas limitações no que diz respeito ao desenvolvimento de sistemas que tenham de processar acontecimentos em tempo real, mesmo no caso do método de análise proposto por Yourdon que se concluiu ser o que melhor se coaduna com a especificação de sistemas deste tipo. No exemplo apresentado este facto foi particularmente significativo na descrição das complexas interacções entre os processos de controlo do sistema e as várias entidades externas, que pelo facto de não estarem sincronizadas, podiam gerar novos pedidos a meio do processamento de pedidos anteriores: como esses acontecimentos não podem ser ignorados, nomeadamente no caso de serem imagens ou avisos de alarmes, podem-se originar situações de recursividade indirecta que não são facilmente detectáveis através da utilização desta técnicas. No entanto parece-nos que estas limitações da análise não são demasiado importantes, dado estes aspectos deverem ser considerados no projecto estruturado e não na análise.

No que diz respeito ao Diagrama de Estrutura criado no projecto estruturado, pareceu-nos bastante útil, fácil de criar e de manter. Estes diagramas são bastante úteis na implementação do sistema dado permitirem definir um modelo síncrono e possibilitarem, por isso, considerar com facilidade diversos aspectos, tais como: procedimentos de recuperação de erros, tratamento de excepções, problemas relacionados com limitações relativas ao ambiente de implementação bem como o problema da recursividade indirecta já referido. No entanto o projecto estruturado contém algumas limitações, nomeadamente, não induzir o programador na implementação a organizar convenientemente o código e os dados.

Tendo por base o objectivo de desenvolver o software de uma forma estruturada por forma a facilitar o seu desenvolvimento, reutilização, manutenção e compreensão, foram usadas com sucesso as seguintes estratégias:

1-Organizar as diferentes rotinas em três tipos principais: rotinas de *Interface*, rotinas de *Processamento* e rotinas de *Comunicação*.

2- Todas as rotinas de um dado tipo devem estar organizadas num só directório específico e por sua vez em cada directório as várias rotinas devem estar organizadas em vários ficheiros existindo um só ficheiro com as rotinas, de mais alto nível, que servem de interface para o exterior.

3- A relação entre as rotinas e os dados que estas manipulam, deve ser clara para quem examinar o software desenvolvido, nomeadamente os dados (variáveis) internos a cada módulo como se pode ver na figura 7.

4- Cada módulo principal deve estar organizado de forma a poder interromper temporariamente um dado processamento mais demorado, preservando toda a informação necessária ao seu recomeço. O recomeço desse ou de outros processamentos temporariamente interrompidos relativos a esse módulo deve poder ser despoletado por uma rotina de *polling* (desse módulo) que a aplicação possa evocar, periodicamente ou quando ocorrer um acontecimento especifico.

5- O processamento de acontecimentos em sistemas de tempo real deve prevêr a chegada de novo(s) acontecimento(s) para processar ainda antes de pedidos anteriores terem sido completamente processados, nomeadamente quando estes últimos têm de aguardar por confirmações vindas de entidades externas para poderem ser terminados.

6 -Cada módulo deve possuir um ficheiro de LOG onde todos os pedidos recebidos e todas as respostas às suas acções fiquem gravadas para consulta posterior.

7- Cada módulo deve ainda ser capaz de permitir à aplicação fazer o seu próprio *reset* para que seja possível numa situação anómala, ou mesmo numa situação normal, voltar ao ponto de partida.

REFERÊNCIAS

[1] J. Santos, F. Ramos, O. Santos, "Sistema de Televigilância suportado na RDIS", Revista do DETUA, Vol. 1, Nº 2, Setembro de 1994, pp 169-180.

[2] T. DeMarco, Structured Analysis and System Specification, Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1979.

[3] C. Gane, T. Sarson, "Structured System Analysis: Tools and Techniques", Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1979.

[4] E. Yourdon, "Modern Structured Analysis", Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1989.

[5] S. McMenamim, J. Palmer, "Essential System Analysis", Yourdon Press, 1984.

[6] P. Ward, S. Meller, "Structured Developemnt for Real-Time Systems", vol. I, II and III, Yourdon Press, 1985.

[7] J. Martin, C. MacClure, "Structured Techniques: Basis for CASE", Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1988.