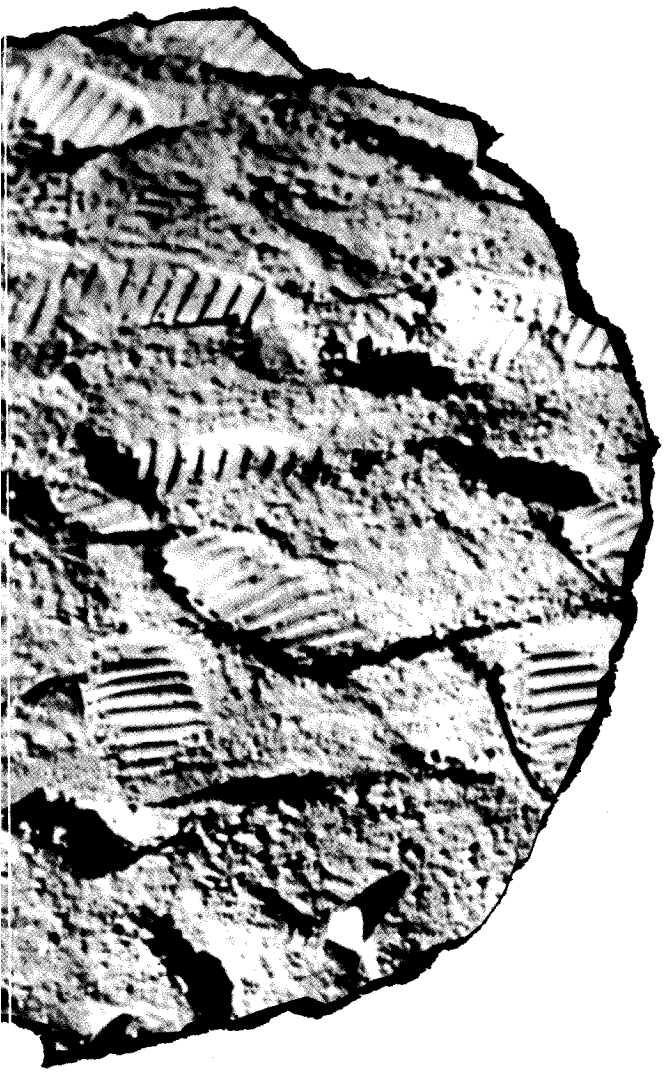
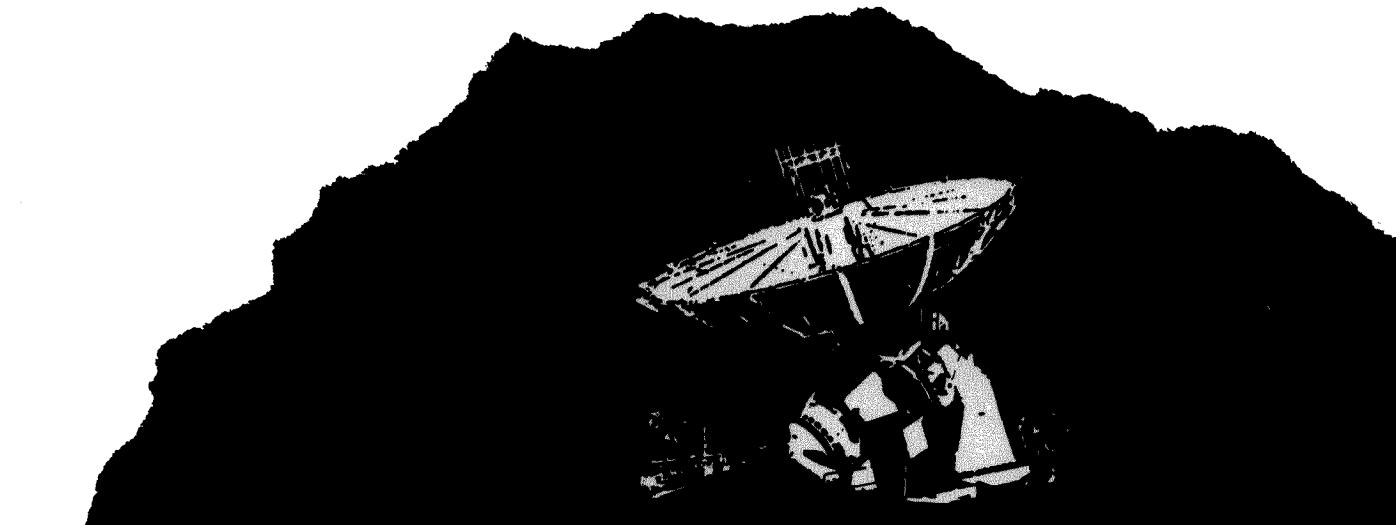
ISSN: 1645-0493

Electrónica e Telecomunicações

universidade de aveiro

theoria poiesis praxis





AVEIRO • SET • 2001 • VOL. 3 • Nº4

Revista do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro

Electrónica e Telecomunicações

Revista do Departamento de
Electrónica e Telecomunicações
da Universidade de Aveiro

Editores:

Francisco Vaz
Augusto Silva

Morada e Secretariado:

Departamento de Electrónica
e Telecomunicações
Universidade de Aveiro
Campo Universitário
3810 AVEIRO
Portugal

Artes Gráficas :

Sérgio Cabaço

Tiragem : 400 exemplares

Depósito Legal Nº 115607

ISSN: 1645-0493

Editorial

Esta edição da Revista do DET surge num momento em que temos concerteza presente algumas das recomendações pertinentes resultantes dos recentes processos de avaliação de algumas licenciaturas ministradas pelo nosso Departamento. Entre essas recomendações é importante salientar o reforço das capacidades de expressão oral e escrita reconhecidamente importantes para exercício da engenharia nas suas múltiplas vertentes.

Sem falsas modéstias e como editores em exercício, cremos que iniciativas de publicação nesta Revista por alunos finalistas e/ou de pós-graduação para além divulgarem actividades de ensino e investigação que se desenvolvem no Departamento terão ainda o mérito fundamental de apelar, ao menos por uma vez, ao exercício correcto das capacidades de expressão escrita por parte dos nossos formandos. Será pois nossa obrigação como responsáveis por formação e investigação reconhecer a qualidade e promover a sua divulgação. No contexto dos seus objectivos e não sendo obviamente a única instância de publicação cremos que a Revista do DET pode continuar a ser um veículo priviligiado de divulgação.

Os editores aproveitam para manifestar abertura para continuar a acolher, ainda que de forma limitada, iniciativas editoriais associadas à publicação de actas de eventos científicos cuja organização envolva de forma significativa membros do DET.

Aguardando desde já contribuições para uma próxima edição, os editores manifestam o seu agradecimento aos autores e a todas as entidades que duma ou doutra forma viabilizaram a concretização de mais uma edição desta Revista.

Francisco Vaz, Augusto Silva

A edição desta revista é subsidiada pela
Fundação para a Ciência e Tecnologia

Índice

[1] Núcleo Regional Norte da Associação Portuguesa de Paralisia Cerebral

[2] Associação Portuguesa de Pais e Amigos da Criança com Deficiência Mental – Núcleo de Aveiro

[3] Dep. de Fisica, Universidade da Beira Interior Covilhã, 6200 Covilhã, Portugal

[4] Escola Superior de Tecnologia de Castelo Branco / DETUA

[5] Instituto Superior de Engenharia de Coimbra / DETUA

Sistema de Atendimento Telefónico Automatizado dos SDUA (SATA-SDUA)

Carlos Rui Gouveia Carvalhal, Arménio Augusto Arvins Pereira Pinto,
Manuel Agostinho de Oliveira Romeiro

Resumo - **Neste artigo apresenta-se um Sistema de Atendimento Telefónico Automatizado, baptizado com a designação SATA-SDUA (Sistema de Atendimento Telefónico Automatizado dos Serviços de Documentação da Universidade de Aveiro), capaz de disponibilizar, via telefone, algumas das funcionalidades acessíveis através do Servidor WWW dos SDUA. Adicionalmente este sistema disponibiliza uma Interface de Gestão, acessível via WWW, que permite ao Administrador configura-lo, geri-lo e obter relatórios da sua utilização. O SATA-SDUA apresenta uma Arquitectura Distribuída baseada no Modelo Cliente-Servidor, permitindo assim que os diferentes processos que o compõem possam ser dispersos por várias máquinas. Foi desenvolvido usando, essencialmente, a Linguagem Java o que o torna portável, económico e de fácil manutenção.**

Abstract – **This paper presents an Automated Phone Attendance System, named SATA-SDUA (Automated Phone Attendance System of the Aveiro University Documentation Services), capable to supply, by telephone, some of the functionalities supplied by the SDUA WWW Server. Otherwise, it disposes an Administration Interface, accessible by WWW, which allows the Administrator to configure it, manage it, and to obtain reports about the uses of it. SATA-SDUA presents a Distributed Architecture based on the Client-Server Model, allowing, in that way, the dispersion of its constituent processes by several computers. It was developed using, essentially, the Java Programming Language that turns it portable, economic, and of easy maintenance.**

I. INTRODUÇÃO

Apesar dos SDUA (Serviços de Documentação da Universidade de Aveiro) disporem de um sistema de atendimento acessível via WWW, que permite aos seus utentes consultarem informação relevante sobre o funcionamento da Biblioteca e realizarem múltiplas tarefas sobre o seu Registo Bibliotecário (Consulta/Renovação de Empréstimos, Criação/Consulta/Anulação de Reservas, Consulta de Empréstimos com Prazo de Entrega ultrapassado, etc.), verifica-se, no entanto, que os seus funcionários são muitas vezes inquiridos, telefonicamente, pelos utentes com questões quer sobre o funcionamento da Biblioteca quer sobre o seu Registo Bibliotecário, o que provoca uma grande sobrecarga sobre os funcionários, em detrimento da qualidade do serviço prestado "ao balcão".

Com o intuito de melhorar esta situação, surgiu a ideia SATA-SDUA (Sistema de Atendimento Telefónico Automatizado dos SDUA), um Sistema de Atendimento Telefónico Automatizado capaz de transpor para o telefone algumas das funcionalidades disponibilizadas pelo Servidor WWW dos SDUA. Este sistema deveria ser tal que:

- A interface com o Utente SDUA fosse implementada por um equipamento terminal telefónico com capacidade de marcação DTMF (Dual Tone Multi-Frequency). Dessa forma a interacção no sentido Utente SDUA→SATA-SDUA seria efectuada através do teclado do telefone e a interacção no sentido contrário através da reprodução de ficheiros áudio.
- Permitisse aos Utentes dos SDUA consultar informação relevante sobre o funcionamento da Biblioteca, assim como também realizar as seguintes operações sobre o seu Registo Bibliotecário:
  - Consultar/Renovar Empréstimos.
  - Consultar/Anular Reservas.
  - Consultar Empréstimos cuja data de devolução tenha sido ultrapassada (Multas).
- Implementasse um sistema de segurança através de mecanismos de autenticação dos utilizadores do Sistema (Administrador SATA-SDUA e Utente SATA-SDUA), assim como também através da definição dos seus papéis (atribuição de permissões).
- Implementasse uma interface de gestão, acessível via WWW, que permitisse ao Administrador SATA-SDUA configurar o Sistema, geri-lo e obter relatórios da sua utilização.

II. ARQUITECTURA DO SISTEMA

O SATA-SDUA apresenta uma arquitectura baseada no Modelo Cliente-Servidor como mostra a Figura 1 que apresenta a sua arquitectura e fluxo de dados. Nesta Figura é possível distinguir um conjunto de processos servidores (Servidor SATA-SDUA e Servidor Administrador SATA-SDUA), um conjunto de processos de suporte aos servidores (Gestor de Comunicação, Gestor da Base de Dados de Administração e Gestor da Base de Dados dos

SDUA), um conjunto de Bases de Dados (Base de Dados de Administração e Base de Dados dos SDUA), geridas por dois dos processos de suporte, e finalmente por um conjunto de processos de interacção com os utilizadores (Cliente Administrador SATA-SDUA e Gestor da Placa Voice-Modem). Estes processos e Bases de Dados podem, por sua vez, serem agrupados, em termos funcionais, em 2 sub-sistemas, o Sub-Sistema do Administrador, que engloba todas as componentes destinadas à gestão do sistema, e o Sub-Sistema do Utente, que engloba todas as componentes envolvidas na interacção (em sentido lato) com o Utente SATA-SDUA, conforme apresentado na Figura 1.

O recurso a esta arquitectura, desde que suportada pela tecnologia apropriada, possibilita a sua implementação num ambiente distribuído, distribuindo, assim, os diferentes blocos funcionais do sistema por várias máquinas, o que permite por um lado a distribuição da carga de processamento e a especialização de cada um dos sistemas computacionais e por outro lado a implementação de níveis de segurança suplementares.

A. Processos Servidores

Servidor SATA-SDUA: Este processo desempenha o papel central do Sub-Sistema do Utente, sendo, dessa forma, o responsável pela gestão e interligação entre os restantes processos que o integram. Sempre que o seu processo Cliente, o processo Gestor de Comunicação, emitir um pedido de autenticação ou de consulta/alteração do registo bibliográfico dum Utente SATA-SDUA, este processo iniciará a negociação com os processos Gestor da Base de Dados dos SDUA e Gestor da Base de Dados de Administração no sentido de satisfazer tal pedido e registar a operação efectuada, respondendo adequadamente ao processo chamador.

Servidor Administrador SATA-SDUA: Processo central do Sub-Sistema do Administrador, sendo o responsável pela autenticação do Administrador SATA-SDUA e satisfação dos pedidos de gestão dele emanados.

B. Processos de Suporte aos Servidores

Gestor de Comunicação: Processo de interface entre o Gestor da Placa de Voice-Modem e o Servidor de Comunicação. Obterá do segundo a informação necessária à satisfação do pedido emitido pelo primeiro e elaborará a resposta que o Gestor da Placa de Voice-Modem apresentará ao Utente SDUA (mensagem de áudio).

Gestor da Base de Dados de Administração: Processo responsável pelo registo, na Base de Dados de Administração, da informação de monitorização de todos os acessos efectuados pelos Utentes SDUA ao sistema.

Gestor da Base de Dados dos SDUA: Este é o Processo responsável pela comunicação com a Base de Dados dos SDUA, implementando dessa forma a interface do sistema com essa Base de Dados exterior ao sistema. Para tal, formulará os quesitos apropriados à obtenção da informação necessária à sua resposta, transmitindo-a de seguida para o processo Servidor SATA-SDUA.

C. Processos de Interacção com os Utilizadores

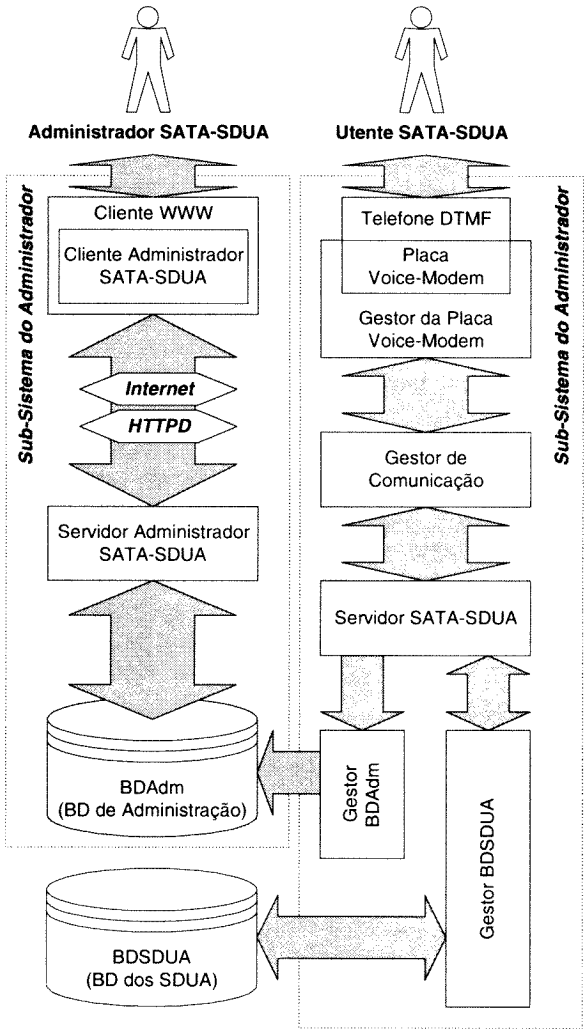
Gestor da Placa Voice-Modem: Processo de interacção com o Utente SATA-SDUA. É o processo responsável pela monitorização, controlo e gestão da placa Voice-Modem e do telefone DTMF que implementa a interface com o Utente SATA-SDUA. Sempre que houver uma interacção por parte do Utente SATA-SDUA comunicará com o processo Gestor de Comunicação no sentido desse pedido encontrar resposta.

Cliente Administrador SATA-SDUA: Processo de interacção com o Administrador SATA-SDUA. Dessa forma, implementará uma interface gráfica amigável, acessível via WWW, e apropriada às operações de gestão a realizar por este, nomeadamente consulta estatística da utilização do sistema e dos registos de sessão (sequência de operações realizadas pelo Utente SATA-SDUA em cada acesso ao sistema).

D. Bases de Dados

Base de Dados de Administração: Base de Dados na qual serão registados os acessos e operações realizadas, em cada sessão, pelos Utentes SATA-SDUA. Esta informação permitirá ao processo Servidor de Administrador SATA-SDUA elaborar relatórios da utilização do sistema, os quais serão úteis, nomeadamente, na resolução de situações problemáticas.

Base de Dados dos SDUA: Base de Dados com a informação sobre o Arquivo Bibliográfico dos SDUA e os seus Utentes. Apesar de estar representada na Figura 1, trata-se duma Base de Dados exterior ao Sistema.



III. OPÇÕES TECNOLÓGICAS

A implementação da arquitectura SATA-SDUA atrás descrita foi precedida de um estudo sobre as diferentes opções tecnológicas disponíveis, por forma a serem escolhidas aquelas que melhor se adaptavam aos objectivos pretendidos. Dessa forma, houve necessidade de escolher uma linguagem de programação, um mecanismo de comunicação entre processos, um mecanismo de comunicação/controlo duma placa de Voice-Modem e um mecanismo de acesso a Bases de Dados Relacionais.

A. Linguagem de Programação

Quanto à Linguagem de Programação a escolha recaiu sobre a Linguagem Java. As razões que levaram a esta escolha são várias: em primeiro lugar, o sistema seria portável; em segundo lugar, a complexidade do desenvolvimento seria menor do que no caso, por exemplo, de C/C++, uma vez que a Linguagem Java é consideravelmente mais simples; finalmente, como os melhores compiladores de Java são gratuitos, seria uma solução mais vantajosa em termos económicos [1]. Dessa forma foi usada a Linguagem Java na implementação de todas as componentes do sistema com excepção da componente de Comunicação/Controlo da Placa de Voice-Modem. A razão para tal prende-se com a indisponibilidade de uma API em Java capaz de comunicar com a Placa Voice Modem. Consequentemente, houve necessidade de dividir a componente de Comunicação/Controlo da Placa de Voice-Modem em dois processos: um desenvolvido em C/C++ que comunica com a Placa Voice-Modem (Gestor da Placa de Voice-Modem), e um desenvolvido em Java que implementa a interface com o resto do sistema (Gestor de Comunicação). [7]

B. Comunicação entre Processos

Os mecanismos de IPC (Inter-Process Communication) considerados foram os seguintes: Sockets, RPC (Remote Procedure Call), CORBA (Common Object Request Brokerage Arquitecture) e JRMI (Java Remote Method Invocation). Como estamos perante um ambiente de programação homogéneo (excluindo o processo Gestor da Placa Voice-Modem que é isolado do resto do sistema pelo processo Gestor de Comunicação) e a informação a trocar é relativamente estruturada (Objectos), Sockets e RPCs foram logo excluídos; ficando a decisão final limitada ao CORBA e ao JRMI. Em termos gerais, CORBA e JRMI são muito idênticos; no entanto, enquanto que o primeiro suporta ambientes de programação heterogéneos (pode ser utilizado na comunicação entre processos que utilizem linguagens de programação diferentes), o segundo está restrito à plataforma Java. Sendo assim, JRMI é muito mais simples de utilizar. Além disso, enquanto que a implementação de JRMI é completamente gratuito, as de CORBA são caras, poucas e incompletas. Dessa forma, a nossa decisão recaiu sobre o JRMI. [2, 3, 4]

O Java RMI é um ambiente de programação distribuído, orientado a objectos e homogéneo. O seu principal objectivo é permitir que os programadores de Java possam desenvolvam aplicações distribuídas, com a mesma sintaxe e semântica das aplicações não-distribuídas.

C. Comunicação/Controlo de uma Placa Voice-Modem

A palavra "consenso" não é a mais apropriada para caracterizar a indústria das telecomunicações; de facto, a competição é tão grande neste campo, que praticamente cada fabricante possui os seus próprios standards para cada uma das áreas existentes. A área da telefonia assistida por computador não foge à regra; os standards que suportam alguma variedade de dispositivos são poucos, mal construídos, e as suas implementações são caras e na maior parte das vezes inacabadas. Mesmo assim, existem

duas grandes soluções: TAPI (Telephony API) e JTAPI (Java Telephony API).

Em termos gerais, a arquitectura da segunda é muito superior à da primeira; isto deve-se não só ao facto de ser mais actual, como também de resultar do esforço de um maior número de fabricantes. No entanto, ainda não existe uma implementação completa da JTAPI, pelo que fomos obrigados a optar pela TAPI [5]. Como esta é uma API de C/C++, a componente de comunicação/controlo da Placa Voice-Modem teve que ser desenvolvida em duas linguagens de programação diferentes, Java e C/C++, tal como já foi atrás referido.

A TAPI é uma API que permite a utilização de serviços de telefonia em computadores pessoais. O termo telefonia refere-se às tecnologias utilizadas para transmitir a voz, de uma pessoa para outra pessoa, ao longo de uma determinada distância. Sendo assim, o objectivo da TAPI é possibilitar que os computadores substituam uma dessas pessoas, ou mesmo as duas. A única implementação que existe foi desenvolvida sob a forma de uma API de C/C++ pela Microsoft, e acompanha todos os sistemas operativos da plataforma MS Windows. A arquitectura da TAPI está representada na Figura 2. [6]

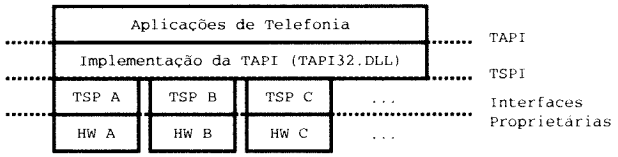


Figura 2: Arquitectura da TAPI.

Os TSPs (Telephony Service Providers), a camada de software imediatamente inferior à TAPI, são, normalmente, fornecidos pelos fabricantes, no entanto, existe um TSP genérico, o UNIMODEM/V, que acompanha os sistema MS Windows e que é capaz de funcionar com uma grande variedade de dispositivos. [6]

Em termos de funcionalidades, a TAPI apresenta as seguintes características:

- Controlo centralizado nas chamadas;
- Acesso ao media através das APIs standard;
- Independência da rede;
- Independência do modelo de ligação;
- Independência da plataforma (sempre que possível);
- Possibilidade de partilha das linhas telefónicas entre diversas aplicações.

D. Acesso a Bases de Dados Relacionais

Em Java existe uma solução de acesso a Base de Dados extremamente fiável e simples de utilizar, a JDBC (Java Database Connectivity). Sendo assim, não foi necessário procurar muito para chegar a uma solução.

A JDBC é uma API orientada a objectos e independente da plataforma, que fornece aos programadores de Java uma solução para o acesso a BDs relacionais. Tal como a maioria das APIs de Java, o objectivo do seu modelo é a simplicidade de utilização. [8]

IV. CONCLUSÕES E TRABALHO FUTURO

O SATA-SDUA, cumpre satisfatoriamente os objectivos propostos, com a mais-valia de apresentar uma Arquitectura Distribuída Cliente-Servidor, o que permite que as suas diferentes componentes sejam dispersas por várias máquinas. O facto de ter sido desenvolvido usando, essencialmente, a linguagem Java torna-o portável, económico e de fácil manutenção. Apesar de actualmente ainda não estar a ser utilizado, acredita-se que tal venha a acontecer num futuro muito próximo, por se tratar dum produto completo, robusto e fiável.

Em termos de trabalho futuro, existem algumas melhorias que poderiam ser implementadas:

- Substituição da TAPI pela JTAPI, assim que houver uma implementação desta.
- Suporte para múltiplas chamadas em simultâneo.
- Parametrização dos diálogos de navegação através do uso de uma linguagem Scripting ou de técnicas de manipulação gráfica.

AGRADECIMENTOS

Aos SDUA, em especial à Dra. Emília Araújo, ao Eng. Filipe Bento e ao Miguel Pinto, pelos meios técnicos disponibilizados para a realização deste trabalho. A eles um muito obrigado.

REFERÊNCIAS

[1] M. and Walrath Campione, The Java Tutorial, 2nd Edition, Sun Microsystems.

[2] Govind Seshadri, Fundamentals of RMI, Java Developer Connection, 2000.

[3] Java Remote Method Invocation, Revision 1.4, Sun Microsystems, 1997.

[4] Java RMI Tutotial, Revision 1.3, Sun Microsystems, 1997.

[5] Java Telephony API, Revision 1.1, Sun Microsystems, 1997.

[6] C. Shells, Windows Telephony Programming, Addison Wesley, 1998.

[7] K. Gregory, Special Edition using Visual C++ 6, Sams.net Publishing, 1998.

[8] JDBC Guide: Getting Started, Sun Microsystems, 1997.

A Solidariedade Social no âmbito do Programa Aveiro – Cidade Digital

Nelson Pacheco da Rocha, Alexandra Queirós, Anabela Mouro, Fernanda Romão, Mónica Almeida, Clara Sousa, Joaquim Alvarelhão[1], Carlos Ventura[3]

[1] Núcleo Regional Norte da Associação Portuguesa de Paralisia Cerebral

[2] Associação Portuguesa de Pais e Amigos da Criança com Deficiência Mental – Núcleo de Aveiro

Resumo - **O artigo apresenta o enquadramento genérico da área de intervenção Solidariedade Social no âmbito do Programa Aveiro – Cidade Digital, os seus objectivos a longo termo e os resultados alcançados durante a primeira fase do Programa.**

Abstract - **The article presents the generic framework of Social Solidarity intervention within the Programme Aveiro – Digital Town, its long-term objectives and the results achieved during the first stage of the Programme.**

I. INTRODUÇÃO

Se, por um lado, a Sociedade da Informação constitui uma oportunidade única para o desenvolvimento económico e social do país, por outro, ela pode constituir igualmente um factor de agravamento do fosso que ainda nos separa dos países mais desenvolvidos. Urge, por isso, procurar as melhores práticas que permitam concretizar a Sociedade da Informação em Portugal. O Programa Aveiro - Cidade Digital procura responder a este importante desafio pela procura das melhores práticas de desenvolvimento e introdução das tecnologias da informação e da comunicação à escala de uma cidade e pela demonstração dos benefícios que elas podem proporcionar. Mais do que disponibilizar infra-estruturas e sistemas, a construção da Cidade Digital passa por uma transformação radical dos hábitos e dos comportamentos dos cidadãos e das Instituições que fazem a cidade. Assim, foi considerada de importância capital que o Programa promovesse a participação voluntária de todos e, simultaneamente, conservasse uma atitude flexível, encorajadora de iniciativas relevantes que nasçam espontaneamente junto de agentes citadinos interessados e tendo sempre presente que o objectivo último do Programa é a melhoria da qualidade de vida na cidade, em todas as suas vertentes. Nesse sentido, foram consideradas diversas áreas estratégicas: i) Construir a Comunidade Digital; ii) Autarquia e Serviços de Âmbito Concelhio; iii) Escola e Comunidade Educativa; iv) Universidade e Comunidade Universitária; v) Serviços de Saúde; vi) Solidariedade Social; vii) Tecido Produtivo; viii) Informação Cultural e de Lazer.

O Programa Aveiro – Cidade Digital, em geral, e a implementação dos objectivos definidos para as várias áreas foi planeado para oito anos, subdivididos em duas fases. A primeira fase, já concluída, foi desenvolvida numa base experimental, contou com uma participação de mais do que 4000 utilizadores e deverá ser seguida por uma segunda fase para a qual se deseja um desenvolvimento sustentado.

II. OBJECTIVOS A LONGO TERMO DA ÁREA DE INTERVENÇÃO SOLIDARIEDADE SOCIAL

A Solidariedade Social, nas suas componentes de segurança social e de inserção de pessoas com necessidades especiais, é uma área de intervenção fundamental para a construção da Cidade Digital. As tecnologias da informação e da comunicação devem contribuir para uma sociedade mais justa em que a igualdade de oportunidades seja efectiva, evitando a introdução de novas barreiras e de novos desajustamentos sociais.

A utilização das tecnologias da informação e da comunicação no apoio a grupos populacionais desfavorecidos, em particular deficientes e idosos, devem dar corpo às novas perspectivas de abordagem das pessoas com necessidades especiais: estas, ao colocarem como questão fundamental a participação social das pessoas desfavorecidas, a normalização das suas vidas e o estudo das suas capacidades, esbatem as fronteiras entre o normal e o patológico, demonstrando que, de um ou de outro modo, qualquer pessoa, num momento determinado da sua vida, pode encontrar-se em situação de desvantagem, que se pode definir como discrepância existente entre as capacidades de um indivíduo e os recursos existentes numa comunidade.

Assim, não é tanto a sua deficiência que o incapacita mas o tipo de interacção que a pessoa estabelece com

determinados elementos do meio envolvente. A componente Solidariedade Social do Aveiro – Cidade Digital baseia-se neste paradigma e tem como questão central a autonomia da pessoa desfavorecida, num ambiente o menos restritivo possível. A aplicação deste paradigma deve traduzir-se, por um lado, em novos programas de intervenção específica conducente a uma maior igualdade de oportunidades tendo em vista atingir a normalização da vida quotidiana e uma maior integração social. Por outro lado, há a necessidade de contribuir para uma maior flexibilidade dos sistemas e serviços, de forma a que estes vão cada vez mais ao encontro das necessidades da população, em geral, e, em particular, das pessoas desfavorecidas. Isto implica uma atenção muito especial aos novos desenvolvimentos tecnológicos que, normalmente, estão associados ao aparecimento de novas barreiras. Obviamente, esta preocupação tem que ser encarada de uma forma abrangente, não só na área de Solidariedade Social, mas em todoas as áreas de intervenção do Aveiro – Cidade Digital. Estes objectivos genéricos foram substanciados num conjunto de metas que serão de seguida enumeradas.

A. Contribuir para uma maior Acessibilidade dos Sistemas e Serviços

A comunidade só será acessível (ou inclusiva) se todos os seus membros a puderem utilizar, isto é, movimentar-se no seu espaço e utilizar os seus serviços e equipamentos sociais com o máximo de autonomia possível.

Nesta perspectiva há todo um conjunto de situações complexas que necessitam ser alteradas e para cujas alterações as novas tecnologias da informação e da comunicação podem contribuir. Neste particular, a utilização e divulgação das tecnologias de apoio podem ter um papel decisivo.

Em Portugal, a forma de acesso à tecnologia de apoio é anacrónico e necessita de ser profundamente repensado a nível nacional. No entanto, alguma coisa pode ser feita a nível regional, nomeadamente: i) Formação dos agentes institucionais e educativos [1]; i) Criação de centros de competência integrados, multidisciplinares e orientados à incapacidade (ie, orientados à substituição, "compensação", prevenção das funções necessárias à execução de determinada actividade) e não à deficiência; iii) Um acompanhamento e aconselhamentos efectivos pós-prescrição (follow-up); iv) Uma componente de avaliação e acompanhamento que pode fornecer dados preciosos, sistematizados e de forma regular aos investigadores da área; v) A criação de um conhecimento sistematizado de problemas e soluções que poderão ser úteis noutras regiões do país.

No entanto, tão importante como remover as barreiras existentes, é ter o cuidado de não criar novas barreiras quando se desenvolverem novos serviços. Dado que o grande motivação da Cidade Digital é o de promover a utilização de novos serviços com um grande suporte tecnológico há a necessidade de uma acção transversal ao Programa que garanta que esses novos serviços não signifiquem novas barreiras para determinados grupos populacionais, para o que deve: i) Consciencializar a sociedade para o problema da acessibilidade; ii) Promover a acessibilidade na Internet; iii) Fomentar o aparecimento nos planos de estudo das escolas tecnológicas de temas relacionados com o Projecto Universal; iv) Apostar na acessibilidade dos novos serviços de carácter digital através da sistematização exaustiva dos requisitos dos utilizadores, criação de uma Provedoria do Excluído como um mecanismo de alerta para sistemas e serviços menos acessíveis e capaz providenciar e disseminar boas práticas.

B. Sistemas de Informação para as Instituições

A Cidade Digital pressupõe também a modernização dos serviços das Instituições de Solidariedade Social, nas suas componentes de facilitação de acesso dos utentes, reorganização dos procedimentos de gestão e administrativos, para além de permitir a provisão de novos serviços. Isso passa não só pela aquisição de equipamentos, mas também pela mudança de atitudes para a qual é necessária financiamento, formação e parcerias estratégicas com entidades possuidoras de competências e saber-fazer que não existem nas Instituições.

Por outro lado, não é fundamentado dizer que não é prioritário introduzir as tecnologias da informação e da comunicação nas Instituições porque há um conjunto de outras necessidades básicas que não estão satisfeitas. Na verdade, sendo o custo dos recursos humanos a componente mais significativa dos orçamentos da Instituições é preciso uma alteração profunda dos procedimentos organizacionais, por forma que os referidos recursos humanos tenham uma maior disponibilidade para a prestação de serviços de qualidade: tudo o que for feito para reduzir as tarefas administrativas e repetitivas é salutar e contribui para aumentar os "momentos de atenção" em relação aos utentes.

É certo que há uma grande necessidade de informatização e implementação de sistemas internos e integrados de informação, não só para a gestão contabilística, mas também com a gestão dos programas de intervenção. Mas, mais do que isso é necessário ter em conta que, decorrendo da sua própria acção de um trabalhador social, ele é, fundamentalmente, um gestor de informação, pelo que as novas tecnologias devem ser utilizadas para introduzir mais objectividade nas diferentes tarefas sociais. No entanto, a alteração de procedimentos não é conseguida pela mera introdução de equipamento. Será necessário: i) Um grande investimento de formação; ii) Criar boas práticas para a introdução das novas tecnologias da informação e da comunicação; iii) Desenvolver parcerias on-line entre as Instituições e entre estas e entidades científicas; iv) Introduzir metodologias de workflow e cooperação remota na realização dos planos educativos individuais (PEIs), desmistificando um pouco a necessidade de um grande número de reuniões nem sempre produtivas; v) Fazer depender das tecnologias

da informação e da comunicação os processos burocráticos, de forma a intensificar a sua utilização generalizada.

A construção de uma Cidade Digital pode também ter um impacto enorme no relacionamento das Instituições com as estruturas locais, regionais e nacionais da Administração Pública: i) Acesso remoto a serviços tão solicitados como o pedido de documentos e preenchimento de declarações por parte de utilizadores finais; ii) Novas formas de relacionamento com as Instituições que intervêm no tecido social local, nomeadamente ao nível da troca de informação e fluxo de documentos; iii) Introdução de metodologias de workflow e cooperação remota no trabalho entre Instituições e a estrutura local da Segurança Social, mais uma vez com o objectivo de minimizar a sobrecarga associada a um sem número de reuniões e, consequentemente, optimizar a produtividade dos recursos humanos; iv) Generalizar a partilha de dados entre as diferentes entidades que intervêm no tecido social, particularmente em respostas que exigem uma estreita cooperação entre os parceiros envolvidos, como são exemplos as intervenção em áreas problemáticas.

C. Promover a Produção de Conteúdos Multimédia Adaptados às Pessoas com Necessidades Especiais

Os conteúdos multimédia podem desempenhar um papel importante no auxílio de pessoas com necessidades especiais, quer para optimizar programas de reabilitação, quer para a aprendizagem de novas capacidades, quer ainda providenciar mecanismos de ajuda como, por exemplo, mecanismos que permitam ultrapassar dificuldades da vida diária [2].

No capítulo dos programas de aprendizagem tem-se notado um interesse crescente na utilização das novas tecnologias e começam a aparecer alguns conteúdos. No entanto, não existindo conteúdos em português, muitas das vezes tenta-se aplicar ou poucos conteúdos em português existentes (por exemplo, conteúdos de programas associados à primeira infância) a grupos populacionais específicos. Tal conduz necessariamente a uma pouca adequação, pelo que o desenvolvimento de conteúdos que satisfaçam os requisitos de determinados grupos populacionais é uma questão extremamente pertinente. Devem ser desenvolvidos conteúdos para programas de orientação e mobilidade (orientação no espaço, adopção das convenções, associação de conceitos e mapeamento do conhecimento cultural com a informação cognitiva), programas de aprendizagem de tarefas do quotidiano (deslocação na casa, deslocação na rua, vestir, comer, comunicar, etc.) e ainda programas adequados à integração no mercado de trabalho.

D. Serviços de Apoio à Distância

As tecnologias da informação e da comunicação podem providenciar centros remotos com um papel relevante no desenvolvimento de estruturas de apoio a idosos, a pessoas portadoras de deficiência que vivem sozinhas, ou a profissionais e famílias vivendo em áreas isoladas, tanto sob o ponto de vista geográfico como social, e que tem a seu cargo utentes ou familiares com qualquer tipo de incapacidade. Isto é particularmente importante, porque Portugal é um país onde há uma falta de profissionais qualificados, uma assimetria marcada entre regiões, em geral, e, em particular, entre algumas áreas rurais e urbanas, relativamente ao acesso e distribuição de cuidados de saúde, educação, serviços de apoio social e recursos [3].

A aplicação do novo paradigma do apoio social pressupõe que existam redes de auxiliares e técnicos com meios eficazes para estabelecer contactos com a Instituição de suporte, nomeadamente durante as prestações de serviços no exterior. No caso particular dos idosos, é também preciso ter em conta a tendência actual da politica portuguesa em que alguns lares de menor qualidade estão a ser substituídos por famílias de acolhimento.

Assim: i) As novas tecnologias da informação e da comunicação podem auxiliar a manter o contacto com a rede de profissionais e com todas as unidades elementares envolvidas no apoio social; ii) Por outro lado, é preciso considerar toda uma série de serviços remotos que podem ser realizados para auxiliarem os utentes finais, sejam eles, por exemplo, idosos, indivíduos portadores de deficiências ou famílias com jovens deficientes ou crianças a seu cargo, obviando, igualmente, situações de isolamento (Segurança, Vigilância a pedido, Aconselhamento e acompanhamento, Resposta a situações de emergência, Monitorização de parâmetros clínicos); iii) Os serviços à distância devem também ser utilizados para fomentar as relações inter-Instituições, sejam elas a cooperação entre Instituições semelhantes (cooperação entre pares ou apoio técnico realizado por centros de recursos ou competências), ou a cooperação entre Instituições complementares como, por exemplo, um serviço de telemedicina para apoio de um lar de idosos; iv) Finalmente, é necessário considerar que a realização destes serviços, pressupõe a existência de estudos prévios que avaliam as expectativas dos profissionais e definam boas práticas para a introdução de serviços de apoio à distância.

E. Soluções de Emprego para Pessoas com Necessidades Especiais

Para a promoção do emprego de populações deficientes é preciso, antes de mais defender substanciais alterações legislativas. Nas áreas tecnológicas vemos nascer empresas com poucos trabalhadores. O regime cooperativo, que poderia ser um instrumento excelente, permite criar cooperativas com um número mínimo de pessoas que não é comportável nas áreas tecnológicas. Por outro lado, é preciso ter em conta que as pessoas portadoras de deficiência, independentemente da sua capacidade produtiva, necessitam, para desempenharem as

suas funções, que um conjunto de necessidades básicas seja satisfeito (por exemplo, ajudas à vida diária).

No contexto do Aveiro – Cidade Digital foi considerado essencial a criação de telecentros multi-serviços [4] que deverão ter o apoio de entidades mediadoras, ("operadores sociais") que, de entre as tarefas que poderão desenvolver, destacam-se: i) a constituição, ou pelo menos o apoio à constituição, e o acompanhamento do funcionamento de eventuais telecentros; ii) Funcionamento como "face visível" dos telecentros e como garantia inicial da qualidade dos serviços, aspecto essencial dada a novidade do teletrabalho e alguma desconfiança manifestada pelas empresas; iii) A constituição de uma carteira de clientes e de actividades, numa perspectiva de prospecção activa e constante, e o desenvolvimento da fidelidade comercial desses clientes; iv) Fornecimento de formação adequada, directamente relacionada com as oportunidades identificadas e focando, designadamente, o uso de novas ferramentas de trabalho, a aquisição de autonomia e a organização do trabalho; v) A ajuda ao teletrabalhador na negociação de contratos de trabalho adequados às necessidades e expectativas do empregador e do empregado; vi) A providência de acompanhamento psicológico para o teletrabalhador quando ela se mostra necessária; vii) O estabelecimento de relações fortes e de confiança com os empregadores.

F. Formação

O sucesso de projectos de desenvolvimento como os do Programa Aveiro – Cidade Digital está dependente de factores como: i) Requisitos mínimos de formação em tecnologias da informação e da comunicação; ii) Capacidade dos intervenientes trabalharem em equipas multidisciplinares; iii) Intervenientes com preocupações sociais e imbuídos de um espírito de cidadania.

Nesse sentido, para além da preparação de um plano de formação intensiva dos técnicos das Instituições, no qual mais do que ensinar processamento de texto ou folhas de cálculo, deve estar orientada para a interiorização de novos paradigmas que podem permitir a resolução de problemas reais, é fundamental que as escolas das áreas paramédicas e sociais incluam nos seus planos de estudos uma grande incidência sobre as potencialidades das novas tecnologias da informação e da comunicação, e a uma formação superior signifique também a capacidade de trabalhar com eficiência numa equipa multidisciplinar.

III. RESULTADOS DA PRIMEIRA FASE

O programa Aveiro – Cidade Digital, em geral, e a implementação das metas da área de intervenção Solidariedade Social, em particular, foram programados para oito anos e subdivididos em duas fases. A primeira fase, com duração de dois anos, já implementada, teve um carácter experimental e deve ser seguida de uma segunda fase de seis anos, a qual deve ter uma grande sustentabilidade. Na primeira fase, a Universidade de Aveiro assumiu, naturalmente, um papel de coordenação da Área de Intervenção relativa à Solidariedade Social, devido, fundamentalmente a uma série de condições únicas: i) Experiência de alguns anos no desenvolvimento de serviços, aplicações e equipamento para pessoas com necessidades especiais, algumas delas funcionando em cenários reais com largas dezenas de utilizadores e durante períodos de tempo alargados; ii) Experiência em lidar com grupos populacionais específicos, nomeadamente indivíduos portadores de deficiência visual, deficiência auditiva, deficiência mental, paralisa cerebral e idosos; iii) Experiência em trabalhar com equipas multidisciplinares envolvendo engenheiros, psicólogos, sociólogos, terapeutas e assistentes sociais; iv) Uma ligação muito forte com diversas Instituições de Solidariedade Social e de reabilitação, quer a nível local, quer a nível nacional; v) Elos estabelecidos com Instituições nacionais e internacionais com investigação reconhecida na área, conseguidos através da participação em diversos projectos de investigação e desenvolvimento de âmbito europeu (nomeadamente dos programas RACE, TIDE, ACTS, Telematics e Horizon); vi) Colaboração internacional com autarquias reconhecidas como exemplares na implementação de serviços de apoio a deficientes e idosos; vii) Possibilidade de criação de equipas multidisciplinares dentro da própria Universidade.

Esta experiência e capacidade da Universidade de Aveiro foi reconhecidamente útil nos quatro projectos que procuraram contribuir para as metas delineadas pelo programa Aveiro - Cidade Digital para a área da Solidariedade Social da área de Intervenção Solidariedade Social: RESEA (Rede de Serviços da APPACDM), NET ALIMENTAR, IST (Integração Social por Teletrabalho) e MEU (Mobilidade no Espaço Urbano).

O RESEA visou sistematizar a introdução das tecnologias da informação e da comunicação ao nível das Instituições de Solidariedade Social. O projecto, ao integrar a formação com a introdução das novas tecnologias, iniciou uma nova dinâmica na APPACDM (Associação Portuguesa de Pais e Amigos da Criança com Deficiência Mental) que promoveu a reorganização dos serviços existentes na Instituição e o aparecimento de novos serviços, em particular a criação de uma rede que permite a resposta a situações de emergência, assegura a telepresença, facilita a comunicação interpessoal e o aconselhamento remoto (comunicação entre lares residenciais e o domicílio do técnico em prevenção). Por último, mas não menos importante, o projecto promoveu a utilização de conteúdos multimédia adaptados a crianças com necessidades especiais, de onde se destacam os de orientação espacial e mobilidade, cores, volume, aprendizagem de tarefas básicas do quotidiano, comportamento cívico e segurança rodoviária.

Ainda em termos de Instituições de Solidariedade Social o projecto NET ALIMENTAR promoveu a criação de novas formas de interacção entre aquelas e o Banco Alimentar, com recurso às tecnologias da informação e da

comunicação, destacando-se uma campanha permanente de recolha de alimentos e recrutamento do voluntariado.

O projecto IST teve como objectivo genérico a utilização das tecnologias da informação e da comunicação para a promoção da qualidade de vida e integração social das pessoas com necessidades especiais. Como resultados do IST destaca-se o ter-se conseguido que um grupo de formandos deficientes motores participassem activamente na sociedade de informação construindo alguns conteúdos multimédia para o Aveiro - Cidade Digital e uma proposta de modelo de teletrabalho que pode ser replicado noutras situações, em particular noutras cidades portuguesas. Tal modelo, pretendeu identificar qual o tipo de tarefas, mais qualificadas ou menos qualificadas, que numa Cidade Digital poderão ser desempenhadas com sucesso por populações marginalizadas, em geral, e deficientes, em particular. Finalmente, houve ainda a oportunidade de analisar o potencial do mercado constituído pelas empresas da região de Aveiro no que respeita à inserção de deficientes pela via do teletrabalho, em particular determinar qual o modelo de teletrabalho percepcionado como mais adequado pelos decisores nas empresas e quais as actividades com maior probabilidade de virem a ser externalizadas pelas empresas, em regime de teletrabalho.

A comunidade deve ser acessível a todos os seus membros. Os deficientes têm o direito de usufruir de todas as estruturas destinadas às pessoas em geral (...) Art.º38º da "Carta para os Anos 80" da Reabilitação Internacional Este é, no fundo, o princípio filosófico que enquadrou o projecto MEU. Com efeito, a comunidade só será acessível (ou inclusiva) se todos os seus membros a puderem utilizar, isto é, movimentar-se no seu espaço e utilizar os seus serviços e equipamentos sociais com o máximo de autonomia possível. A ideia subjacente ao projecto MEU foi criar um percurso de transportes públicos que demonstrasse as boas práticas em termos de disponibilização de informação acessível a todos os cidadãos e de interface com outros subsistemas de transporte. Por outro lado, o projecto MEU promoveu o exercício de cidadania através da Internet, criando um serviço que permite que o cidadão em geral possa emitir as suas opiniões, os seus elogios e críticas sobre a acessibilidade dos locais públicos (repartições públicas, escolas, Instituições, bancos, hospitais, etc.). A disponibilização dessa informação permite não só o realce dos bons exemplos e a correcção de situações menos acessíveis, mas também que o cidadão deficiente possa saber, à priori, qual a acessibilidade dos locais ou serviços que pretende utilizar.

As sinergias criadas entre os diferentes parceiros e projectos da área de Solidariedade Social promoveram o aparecimento de uma parceria para o futuro que é uma manifestação inequívoca de um desejo de continuar um trabalho em comum, por forma a criar uma massa crítica e promover uma verdadeira Sociedade de Informação inclusiva. Esta parceria para o futuro tem já visibilidade no portal Reabilitação Rede de Recursos, que pretende ser o ponto integrador de informação sobre reabilitação. Este portal, ainda numa fase embrionária que, entre outras coisas, contém uma biblioteca digital e um centro distribuído de informação sobre ajudas técnicas.

Paradigmaticamente todas as tecnologias que permitem a integração e disseminação do conhecimento estão disponíveis. Por outro lado, há muito conhecimento disperso e muita experiência acumulada, nomeadamente resultante de projectos já desenvolvidos, relativamente aos quais não foi, por vezes, possível tirar as ilações devidas. O que falta são metodologias capazes de sistematizar e divulgar todo esse conhecimento e combater um problema cultural endémico português: a extrema dificuldade em sistematizar os conhecimentos e as experiências, positivas ou negativas, e partilhá-los com os outros. A biblioteca digital entretanto criada e que continua em desenvolvimento, pretende contribuir para uma política de conteúdos portugueses e contém já um conjunto de obras relevantes para a Solidariedade Social, estudos de casos, sistematização de experiências relevantes. a disseminação de boas práticas, uma colectânea de legislação on-line actualizada e comentada, e disponibilização de informações pertinentes sobre todas as Instituições da região, para o que é necessário uma atitude colaborante dessas mesmas Instituições, nomeadamente ao nível da actualização da informação.

Existem necessidades crescentes em obter informação útil e sistematizada sobre tecnologias de apoio, necessidadese essas que vão desde a pessoa individual, que não sabe o que é mais adequado para colmatar a sua capacidade, ou mesmo a quem se dirigir para obter informação; às Instituições de Solidariedade Social, que se deparam com uma imensidão de ajudas técnicas e bastante específicas, sem terem recurso a uma informação descritiva do que existe, qual a sua classificação em termos da norma ISO 9999 e onde pode ser comprado; ao próprio Estado com orçamentos relativamente elevados para ajudas técnicas e que não sabe onde são gastos, quem prescreveu e se, no final, o utente consegue obter a ajuda técnica mais adequada ao seu problema e pelo período de tempo que vai realmente necessitar daquela; até aos próprios fornecedores que muitas vezes comercializam ajudas técnicas sem o saberem. Nesse sentido, para além de um grande esforço de consciencialização para a potencialidade das tecnologias de informação e da comunicação, iniciou-se a implementação de uma rede de recuros de tecnologias de apoio que possibilita a consciencialização para o papel da sociedade de informação, o desenvolvimento de soluções integradas (mais do que a ajuda técnica per si é necessário a definição de solução integradas e adequadas a cada caso específico), eliminar uma visão sectorial (o papel de uma ajuda técnica não é necessariamente diferente numa situação de emprego, numa situação de educação, ou dentro de uma Instituição), uma ligação à Universidade para uma rápida e eficiente transferência de soluções inovadoras e a criação de um conhecimento sistematizado de problemas e soluções que poderão ser úteis noutras regiões do país.

IV. CONCLUSÕES

Um das metas principais da fase experimental do Aveiro - Cidade Digital foi alcançada: a cidade está consciente de que as tecnologias da informação e da comunicação podem significar uma revolução metodológica e permitem, só por si, que a pessoa portadora de deficiência ultrapasse algumas das suas incapacidades. Por outro lado, deve ser também tido em conta que a cadeia de confiança entretanto criada é tão importante como os novos serviços desenvolvidos e terá necessariamente implicações na implementação da segunda fase do Programa.

V. REFERÊNCIAS

[1] Rocha, N., et al., "A Distance Training and Telework Experiment for People with Cerebral Palsy", Assistive Technology on the Threshold of the New Millenium (Christian Buhler and Harry Knops, ed.), IOS Press, 1999.

[2] Montgomery, A., et al., "Identifying Computer Assisted Instruction for People with Severe Intellectual Disabilities: A literature review", European Journal on Mental Disability, vol. 3, N.9, pages 32 to 46, 1996.

[3] Pereira, L.M., et al., "Distance Support and Elderly People: Overview on Three Projects", Assistive Technology on the Threshold of the New Millenium (Christian Buhler and Harry Knops, ed.), IOS Press, 1999.

[4] Santana, S., et al., "Teletrabalho e Incapacidade: Análise de um Inquérito a Empresas Portuguesas", a ser publicado na Revista Portuguesa de Gestão.

Building the Wireless Universe

EU Research on the Move

José Fernandes, Fabrizio Sestini, João Schwarz da Silva[1]
Mobile and personal communications and systems, including satellite systems and services
European Commission DG Information Society, Brussels

Resumo **- Este artigo apresenta uma visão geral das actividades em curso do programa da União Europeia para a Sociedade da Informação (IST) na área das comunicações moveis, acesso sem fios e sistemas via satélite. As linhas de orientação da investigação especificas desta área são apresentadas em detalhe, com referencia aos projectos já em curso, definindo um possível percurso para evolução das comunicações moveis para além do UMTS.**

Abstract **- This paper provides an overview of the activities currently carried out under the IST research programme of the European Commission, in the area of mobile and wireless communications and satellite systems. The research orientations specific to wireless communications are described in detail, with reference to the projects already approved in each area, defining a possible roadmap for the evolution of Mobile Communication systems beyond UMTS.**

I. INTRODUÇÃO

A. The 2G Success Story

Mobile telecommunications is playing a crucial role in today's economy and lifestyles, as witnessed by the impressive statistics on its diffusion. Amongst the four largest cellular markets, including China, USA and Japan, Europe is now at the first place, with a very steep increase which is expected to be sustained over the next few years. In May 2001 the number of GSM subscriptions exceeded 70% of the EU population, Figure 1. The yearly subscription growth in some EU countries, such as Spain or Germany, is in excess of 100% and the annual penetration increments, since 1995, are quite extraordinary, Figure 2. This may be attributed in large part to the multiple benefits generated by the single second generation (2G) digital standard agreed for Europe as opposed to the situations in other world regions where multiple technological standards were advocated. Among them is the possibility of seamless service provision over networks managed by different operators, the economies of scale both for users and network operators, the international roaming that enables the use of the same handset over all European countries and in many regions abroad, and on top of all the effective competition between operators which has led to increasing benefits for the customer in terms of tariffs and quality of service. The GSM standard, now in use by over half a billion subscribers worldwide deployed in more than 168 countries, is indeed a perfect illustration of the rewards possible through an articulated European policy in telecommunications such as that carried out in the past 10 years.

As the penetration of GSM is approaching 100% and reaching the network saturation in certain areas, operators and manufacturers are preparing the ground for the commercial start of the third generation (3G) of mobile communications, which brings the promise to allow convenient mobile access to the Internet with all foreseeable and non-foreseeable implications (m-commerce, m-networking, etc.).

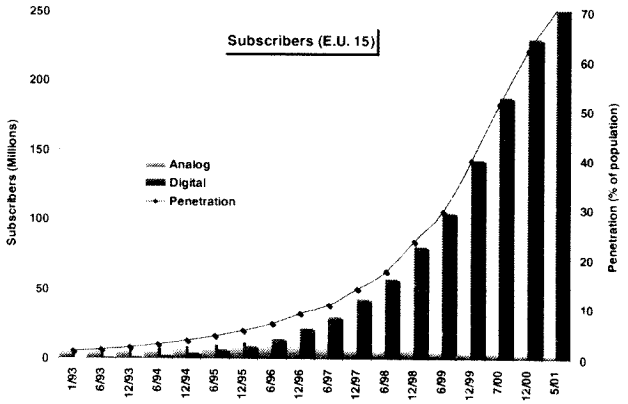


Figure 1 – Number of subscribers in EU, in absolute terms and as a percentage of the population

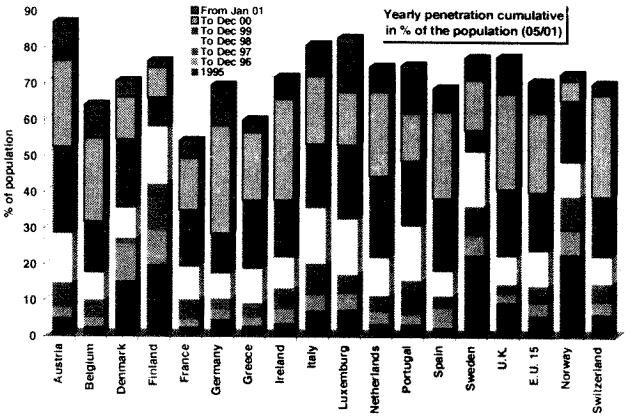


Figure 2 – Cumulative trend of yearly penetration in European countries, in percentage of each country's population

B. The 3G Ambition

[1] *The views expressed herein are those of the authors, and do not necessarily reflect the views of the European Commission.*

European efforts in 3G, referred to as UMTS, followed a process very similar to the one that led to GSM, in a time frame shifted by nearly 10 years. Indeed, the first UMTS R&D project was launched in 1989, with spectrum being first identified for this purpose by WRC'92. Many EU-funded R&D projects under the RACE and ACTS programs have contributed to lay the foundations of the UMTS standard (as for network aspects and tools, services, radio aspects), and have in particular made significant contributions towards solving the most controversial issue – the choice of the air interface– which was made by ETSI in January 1998. The UMTS vision, initially developed in the context of research and development activities, is nowadays being further developed and refined by the 3GPP. The European proposal together with other air-interface proposals are members of the IMT 2000 family endorsed by WRC 2000. In parallel to the standards and R&D work, the European Commission very early on launched a process aiming at setting the licensing and regulatory framework to be used for UMTS. This process culminated in the UMTS Decision which inter-alia requires Member States to allow for the deployment of commercial UMTS networks by 2002. As demonstrated by the amounts that operators have been willing to pay for UMTS licenses, the expectations placed on mobile internet are of unprecedented proportions and will no doubt place Europe at the heart of the coming wireless revoluton.

C. Beyond 3G

While the mobile telecommunications industry is currently considering how best to reap the benefits of the UMTS "always-on" technology, that will ultimately provide the users with 2 Mbit/s, researchers world-wide have started considering solutions beyond third generation, trying to fulfil the expectations of the latest self-realising prophecy of this turn of millennium, after the Moore's law: a new generation of mobile systems every 10 years. Today's society is indeed at the threshold of a communications revolution, of extremely large proportions, where wireless will become the dominant "life-style", characterised by a proliferation of terminals and devices permitting new forms of untethered communications and the development of a truly personal communications space. While there is a general agreement that wireless communications will play a central role in the future, questions are being asked as to what to expect from fourth generation systems (also better named "future generation"): is it just a matter of higher bit rates and interconnection capabilities, a network of self organised wireless devices (sensors, actuators), or a whole new concept of "mobility", and a new service paradigm?

A short term technological perspective of the likely evolution of a number of different wireless technologies is provided in Figure 3. In order to exploit with a better

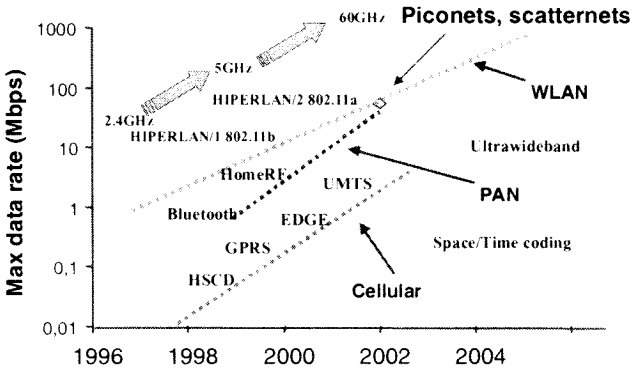


Figure 3 – Short-term Technological perspectives

cost/efficiency higher and higher spectrum regions, two particular disruptive technologies, namely space-time coding and ultra wideband modulation, may bring about major changes in the economic viability of many systems and networks.

But what will the user terminal look like? In the most likely view several different levels of functionality will be possible, corresponding to different sizes –and prices– of the user terminals. What is even more innovative is the concept of "personal mobility": there should be no need to bring around any bulky piece of equipment, as far as the surrounding environment has some kind of "*embedded intelligence*" which can recognise our presence and our preferences, and seamlessly adapt to our needs. The "handset" will only contain the "minimal" set of functionality according to individual requirements, while the additional capabilities will be simply downloaded from the network to reconfigure the terminal itself, or they will be made possible by external "co-operative" devices (e.g. a Bluetooth networked printer, a bigger screen, etc.). An area of particular interest, depicted in Figure 4, is that of the networking of a variety of wireless devices, sensors or actuators, which once embedded in our environment can spontaneously interact with each other and therefore extend our personal communications space.

On the other hand, providers of services will use a variety of radio interfaces to reach the user, depending on what is available or most convenient at a given time and place. An individual channel may be required for a personal video call, but the newspaper or the local map may be simply downloaded in fractions of a second when passing

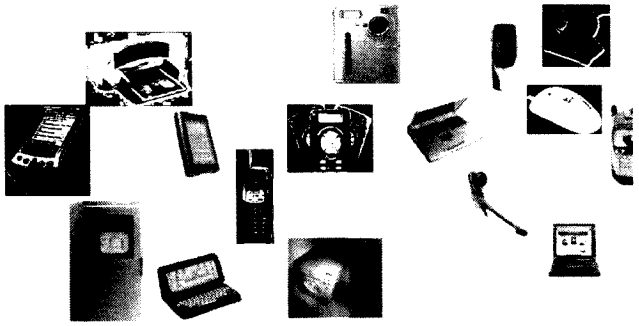


Figure 4 – Self-configured ad-hoc networking of devices

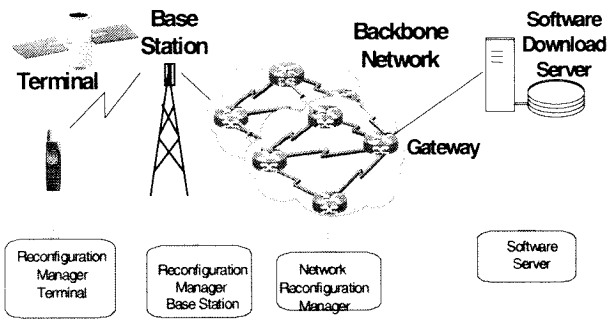


Figure 5– Re-configuration at all network levels

by the cheapest and nearest "e-news kiosk" through a hot spot wireless LAN providing a short-range broadband channel, etc. The terminal needed to do all this could range from a sophisticated re-configurable radio (with a large screen –flat or in the eyeglasses– and adequate input devices) down to the minimal secure "identification tag" (artificial as an RFID tag or natural as a retina) allowing the ubiquitous "network" to recognise our preferences (and billing account) and therefore re-configure itself accordingly. This scenario is depicted in Figure 5, which illustrates the need to reconfigure all network elements, and also to provide a software downloading capability, to cope with specific user requirements.

Thus, any information appliance (resembling a TV, a phone, a web browser, a block notes or whatever else) is just a part of the furniture or of the normal clothing, connected to any other such appliance in order to share information and increase power and functionality. Realisation of this vision presents many societal and technical challenges, including issues of privacy, "digital divide" (i.e. the risk of increased segregation between people who can afford the benefits of the Information Society and those who cannot), security, spectrum allocation, standardisation and interoperability. It requires also a strong linkage and proper articulation between technology, applications, policy developments and implementation. Indeed this concept of "ambient intelligence" is the vision at the basis of the Information Society Technologies (IST) programme.

D. The IST Programme

This IST programme (which is part of the 5th Framework Programme of EU RTD, see www.cordis.lu/ist), is the European Union framework under which the above-referred R&D activities take place. It is implemented through a series of annual workprogrammes, each of which is developed in close co-operation with industry, academia and user organisations.

The IST Programme is structured around four inter-related Key Actions (KAs) all geared towards the achievement of the Programme vision. These are: KA I *systems and services for the citizen*; KA II *new methods of work and electronic commerce*; KA III *multimedia content and tools*, and KA IV *essential technologies and infrastructures*. For the purposes of the workprogramme, the KAs are sub-divided into Action Lines (ALs). "*Mobile and personal communications and systems, including satellite-related systems and services*" are considered as part of Key Action 4. In this area the following specific action lines have been defined:

1. Re-configurable radio systems & networks
2. Terrestrial wireless systems and networks
3. Satellite systems and services
4. Fourth Generation system and network concepts for wireless communications
5. Mobile and personal communications and systems – take-up measures

As the definition of such Action Lines is subject to change and refinement each year, based on the results of the calls for proposals and on the shift in strategic focus in the research arena, the text of the Action Lines for the Calls for Proposals open in calendar year is published in the yearly workprogramme (available through the IST web site http://www.cordis.lu/ist/).

In the remaining of this paper a basic description of such Action Lines will be provided, more in terms of concrete projects which have been already accepted for funding than in abstract words. The definition of the project acronyms (referred to in capital letters) is reported in Table 4, and further details on every single project can be found at www.cordis.lu/ist/ka4/mobile/projects.html. It has to be noted that, since many of the research projects mentioned actually address a wide variety of objectives, their collocation under a certain action line is purely indicative of their main area of activity.

II. SPECIFIC R&D ACTION LINES

A. Re-configurable radio systems & networks

The objective of this Action Line is to create open architectures and service/application development environments that allow heterogeneous wireless networks and platforms to inter-work and adjust adaptively to traffic load and characteristics, services and user requirements. This addresses both terminals and base stations, and includes in particular multi-streaming, dynamic bandwidth allocation, and dynamic spectrum sharing. A key goal for future research is the development of novel techniques and technologies in the area of combined DSP/RF design for efficient and cost-effective adaptive transceivers. Particular interest lies with the combinations of dedicated ASIC implementations and re-configurable logic as well as optimised algorithmic partitioning.

Many different projects are running in this area (Table 1), launched at the beginning of 2000 following the first IST call. CAST, PASTORAL, SODERA and TRUST are focusing on the intelligent and adaptable configuration and management of the radio resources at the physical layer. The feasibility of the selected techniques and radio architecture will also be validated, possibly with the more advanced technologies presently under development (BICMOS-SiGe, SOI & Micro-Machining). PASTORAL is studying reconfigurability already for the GSM (Phase II), W-CDMA (FDD) and TD-CDMA standards, while TRUST addresses the development of re-configurable

terminal platforms from the "user friendly prospective", also including the underlying enabling technologies.

Adaptable reconfiguration of radio resources	*CAST, PASTORAL, SODERA, TRUST*
Open architectures for converging network services (in dynamic usage conditions)	*MOBIVAS, MONASIDRE*
Jointly optimised DSP/RF design for high bit rate modem architectures	*WIND-FLEX*

Table 1 – Projects in AL 4.5.1 Reconfigurable Radio

Upper in the OSI stack, the development of architectural approaches and prototypical implementations of integrated software platforms and systems, adaptable to converging network services and technologies (following TINA/CORBA principles to support a required QoS) is the objective of MOBIVAS, in order to open new opportunities for advanced Value Added Service Providers. The open and adaptive services which are expected should be able to operate in dynamic usage conditions while maintaining the highest available level of quality. Also MONASIDRE is addressing radio resource control (network planning) and QoS management in all IP-based heterogeneous networks, taking into account the flexibility provided by software radio.

More on a system level, WIND-FLEX is studying a high bit-rate flexible and configurable modem architecture, which works in single-hop, ad hoc networks and can provide a wireless access to the Internet in an indoor environment where slow mobility is required. The trade-off between performance and complexity is attained by using a jointly optimized adaptive system (configurable in the run time) which includes the multiple access method, diversity, modulation, coding, equalization and decoding. Bit rates from 64 Kbit/s up to 100 Mbit/s (variable depending on the user needs and channel conditions) are considered in the band of 17 GHz - 19 GHz (5 GHz will be also analysed). Flexibility is attained by using a multicarrier modulation method.

B. Terrestrial wireless systems and networks

The objective of this AL is the study, development and validation of novel terrestrial wireless systems and networks, including fixed wireless access systems and advanced public/private wireless local area networks. It extends to the interworking of mobile/broadcasting systems supporting the provision of broadband multimedia services for interactive and distributive services. Key aspects concern:

- Innovative issues of Quality of Service evaluation and management for multiple radio environments, support of mobility, radio resource control mechanisms (e.g. load sensing) and protocols, including intra-/inter-network handover and seamless roaming between public and private networks supporting full service mobility.
- Optimisation of network elements and terminal performance in heterogeneous (e.g. mobile/broadcasting, public/private) contexts with a variety of symmetric, asymmetric and broadcast services.
- Roaming of location-based services.
- Further enhancement of mobile-related IPv6 capabilities for improved addressing and security.
- The study of novel concepts and architectures for systems and networks offering significant advances in terms of performance, cost, dynamic spectrum sharing, service capabilities and network management features, including high altitude platforms.
- Technological and algorithmic development and demonstration with the objective of enhancing network performance.

Wireless broadband IP(v6)-based network architectures for mobile/fixed IP convergence	*BRAIN/MIND, DRIVE, MCP, MOBY DICK, ARROWS, WINE*
Mobile/broadcasting interworking (FWA to asymmetric services):	*ADAMAS, EMBRACE, MAMBO*
Location and QoS-aware services	*WINE GLASS, CELLO, EMILY*
Network aspects	*ANTIUM, CAUTION, OBANET*
Security aspects	*SHAMAN*
Industrial environments	*R-FIELDBUS*
New transceiver concepts (space-time processing, beamforming)	*ASILUM*
Smart antennas (MIMO channel modelling)	*METRA, SATURN*

Table 2 – Projects in AL 4.5.2 Terrestrial Systems

A growing number of projects in this area (Table 2) is dealing with wireless broadband IP based networks, in order to evolve third generation mobile and wireless infrastructures further towards the Internet. BRAIN and MIND proposes an open architecture allowing for the convergence of fixed Internet, emerging wireless/mobile Internet specifications and UMTS/ GSM/ GPRS, to provide end-to-end IP integrated services. The access network will provide hot spot coverage in urban, suburban, in-building and home cells. Its radio interface, based on HIPERLAN 2, provides the high-speed coverage with data rates up to 20 Mbit/s for the user.

The vehicular environment is the target for DRIVE and MCP which, in different ways, integrate also broadcast networks, such as DAB and DVB-T, to enable spectrum-efficient high-quality wireless IP for universal access to information and support for education, communication, navigation and entertainment. They address the convergence and interworking of cellular and broadcast networks in a common frequency range with dynamic spectrum allocation, and the co-operation between network elements and applications in an adaptive manner. Their objective is to implement IPv6-based mobile infrastructures, handling the available network access taking into account the asymmetry of the communication, the position of the user, the terminal decoding capacity and the user preferences in terms of delay and quality.

MOBY DICK is considering an IPv6-based mobility-enabled end-to-end QoS architecture starting from the current IETF's QoS models and Mobile-IPv6. This will be demonstrated in a testbed of interactive and distributed multimedia applications comprising UMTS, Wireless LANs and Ethernet. ARROWS addresses radio resource control and QoS management for UTRA, to support multimedia IP services up to 2 Mbit/s. Finally WINE approaches the issue from the network side, realising a Wireless-IP adaptation layer configured to optimise different wireless media, and testing it in Bluetooth, IEEE 802.11 and HIPERLAN environments.

Three projects are specifically dealing with the integration/interworking of mobile and broadcasting systems, for fixed wireless access to asymmetric services. While ADAMAS is studying a novel adaptive (at physical and DLC layers) OFDM point-to-multipoint system with bit rates ranging from 64 kbit/s up to 25 Mbit/s (depending on channel conditions or traffic requirements), EMBRACE aims at low cost radio access using MPEG-2 in the downlink and MF-TDMA in the uplink, providing also solutions for nomadic users. Efficient utilization of radio frequency bands is a concern for both projects; in particular the first one will investigate a wide range of service symmetry (from broadcasting to symmetric services). Finally, MAMBO will study the optimisation of the bandwidth allocation for interactive DVB/IP services, based on the perceived QoS. The return channel used for such a feedback can be implemented through LMDS, GSM or UMTS.

Enabling location- and QoS- aware services for wireless mobile users is the goal of WINE GLASS, which will exploit enhanced and/or new IP-based techniques in a wireless Internet architecture incorporating UMTS and WLANs. The Project will develop a testbed incorporating an IP backbone, UTRAN access to IP-based core network, and WLAN access to intranets. Also CELLO and EMILY are exploiting the location capability of mobile terminals, respectively to enhance 2G/3G mobile networks (e.g. with location-aided handover algorithms, or mobility functions supporting multiple wireless systems, including WLANs) and to implement location services integrating terrestrial (E-OTD) and satellite (GNSS) location data in the context of UMTS handset based solutions.

Three projects are dealing with network aspects. CAUTION addresses the development of algorithms and new techniques to provide enhanced capacity management in present (GSM, WAP) and future (HSCSD, GPRS, EDGE) cellular networks. It will provide facilities to monitor the network from a central site so to have an overall view of the network and its components. ANTIUM deals with radio monitoring techniques (mobile network planning) for EDGE/UMTS/DVB-T systems. OBANET addresses specific coverage area management strategies for the optimisation of QoS and spectrum resources in fixed and mobile Wireless Access Networks at 42 / 60 GHz, to be realized through adaptive antennas using photonic beamformers.

Security architectures have been addressed by SHAMAN, in relation to the global roaming in UMTS networks (for instance, through WLAN and Bluetooth) and the dynamic configuration of the components of mobile (e.g. wearable) terminals. Enabling technologies considered are public key infrastructures and smart card security modules.

Devoted to the needs of industrial application services, R-FIELDBUS is developing an innovative high-performance radio fieldbus. Such a radio fieldbus architecture will avoid the need for re-cabling or allow to install new, and probably moving, sensors and control units in manufacturing plants. It will also provide full transparent access to a wide range of information needed on-site, such as data concerning real-time control and status information, or other industrial-type multi-media information with a user-defined QoS. The architecture will be based on the integration of emerging wireless technologies for broadband systems and networks with existent industrial communication protocols (such as those specified in the European Standard EN50170).

Dealing with enabling technologies for mobile communications, the objective of the ASILUM project is to validate new transceiver concepts, for both base station and mobile terminal, to increase the capacity of the future generation of UMTS through new and efficient interference mitigation schemes, based on space-time processing, downlink beamforming, advanced coding and hybrid analog/digital signal processing. A software-based simulation platform running a number of competitive techniques will be designed, implemented and validated to evaluate the performance of the schemes in terms of capacity.

Also in this area is worth mentioning two projects which will analyse the feasibility and evaluate the performance of introducing multi-element adaptive (smart) antennas into mobile terminals in combination with adaptive base station antenna arrays, including transmit diversity, for 3rd generation mobile communication systems (UMTS) and wireless LAN/WANs. They will perform multiple-input and multiple-output (MIMO) matrix radio channel measurement in various mobile communication scenarios. The expected impact of METRA is on standardisation of future phases of EDGE/UMTS, while SATURN will also investigate the potential of array processing to provide enhanced location information for UMTS outdoor broadband networks, particularly in areas where triangulation between several base stations is problematic.

C. Satellite systems and services

The objective of this AL is to study, develop and validate technologies and architectures for the support of multimedia services in the context of advanced mobile systems and of next generation interactive broadcasting systems. The focus is on the:

- technologies and architectures demonstrating a viable implementation of 3G MSS systems, integrating satellite with terrestrial UMTS networks. Fostering convergence between satellite mobile and broadcasting (e.g. S-DAB or S-DVB), it aims at an efficient implementation of

IPv6 multicasting and seamless roaming/service provision across mixed satellite-terrestrial networks.
- Development of advanced technologies and architectures allowing for introduction of novel BSS sub-systems (evolved from legacy broadcasting systems, in integration with terrestrial networks) with scalable support for multimedia services. The work relates for instance to optimised multicasting, caching architectures, dynamic optimisation of spectrum and network resources.

The work is also conceived with the objective of complementing support actions sponsored by Space Agencies (e.g the ESA), without overlapping with it.

Several projects (see Table 3) are dealing with high-speed multimedia via satellite. The objective of BRAHMS is to define a universal user access interface for broadband multimedia (IP) satellite services which is open to different system implementations, including GEO and LEO constellations. It addresses a range of user groups with data rate requirements up to 150Mbit/s. The goal is to open up the market for satellite systems in general, by harmonising the majority of common satellite access network functions whilst allowing flexibility for optimised or proprietary air interfaces to satellite systems. This commonality and flexibility (e.g. for frequency, access type, orbit) is obtained by separating physically-related functions from common service and access functions.

High speed multimedia (IP) via satellite	*BRAHMS, VIRTUOUS, SUITED, SATIN-0, FUTURE*
Convergence of satellite, mobile and broadcasting using IPv6 multicasting	*IBIS, MOBILITY, SATIN-7*
Fixed Satellite Services (multicasting):	*GEOCAST*
Multibeam Ka antennas	*MULTIKARA*
Navigation/communications system integrating S-UMTS and Galileo	*GAUSS*

Table 3 – Projects in AL 4.5.3 Satellite Systems

Also VIRTUOUS aims to design, develop and implement (in test beds) an URAN (UMTS Radio Access Network) Radio Technology Independent part and two URAN Radio Technology Dependent parts able to handle a terrestrial and a satellite link, respectively. It will also design and implement appropriate terminal and network InterWorking Units (IWUs) integrating the GPRS and UMTS segments. The demonstrator will then include three segments: GPRS, terrestrial UMTS and satellite UMTS, and will be further complemented to perform trials of meaningful UMTS service (such as voice over IP).

SUITED aims to design and develop IP based mobile networks consisting of both satellite and terrestrial (UMTS, GPRS, W-LAN) components. Theoretical analysis and experimental work will be performed concerning issues of network architecture (splitting of functions between edge and core network), quality of service and mobility management. The project will carry out extensive series of trials, using an integrated test-bed comprising a multi-segment infrastructure and a multi-mode mobile terminal, capable of operating seamlessly with both satellite and terrestrial networks.

SATIN-0 and FUTURE study new S-UMTS architectures for integration in the UMTS core network, to provide IP based, point-to-multipoint services with end-to-end QoS functions.

Some of the projects selected in the third IST call are addressing the convergence of satellite, mobile and broadcasting using IPv6 multicasting. In this context IBIS will design, develop and test a satellite interactive system (combining the DVB-RCS and DVB-S), to support Interactive TV, Internet and Multimedia services, interworking with the terrestrial network. With a more focused scope, MOBILITY is addressing the provision of mobile satellite TV through DVB-S for mobile (maritime, car, etc.) users, while SATIN-7 deals with the set-up of high-speed interactive IP and multicast services for info-kiosks using DVB and VSAT as a return channel.

In the field of pure FSS systems, GEOCAST (launched in the more general framework of KA4) is investigating the use of geostationary satellites used for TV broadcasts as a platform for two-way IP-based data services (multicasting). GEOCAST intends to help the definition and standardisation of next generation multicast systems, which are evolving towards higher frequencies (Ka and EHF) and more complex missions (multi beams, inter satellites links, regenerative payloads with on-board processing).

Not directly targeting system aspects, MULTIKARA will design and test innovative multibeam receiving antenna around 30 GHz with its associated microwave circuits and evaluate its feasibility for future in-flight use. The ultimate goal is to provide an integrated Tx/Rx Ka band antenna for high rate communications via satellites, complying with constraints of user equipment, such as small sized antennae and low power transmitters, in the Ka band (18-31 GHz), the only non-saturated one available.

Belonging to the area "Intelligent transport infrastructure and mobility management", GAUSS aims at defining an integrated navigation/communi-cations system using S-UMTS as the communications component of Galileo. Such a project will also develop applications in the field of health care and waterways navigation.

D. Fourth Generation system and network concepts for wireless communications

With a long-term perspective, the objective of this AL, which has been open only in June 2000, is to prepare the ground for the likely technological and service evolution from current cellular and wireless systems and networks. Key goals are:
- to investigate advanced and innovative concepts such as self-aware, self-organising ad-hoc wireless networks;
- to develop innovative air interfaces for scalable pervasive connectivity;
- to assess potential spectrum requirements and co-existence issues, including the study of strategies and the development of appropriate tools allowing a distributed

flexible management of the spectrum resources;
- smart interfaces and innovative applications including service personalisation and global portability.

The scope of any future work in the area of "4th generation" was refined in the workprogramme 2002 on the basis of the successful proposals selected so far. Some projects will start soon their activities. However, some projects in other IST sectors (such as the *Future and Emerging Technologies* longer-term research area) have already been selected which are addressing related issues. Several projects in such an area are differently interpreting the vision of "ambient intelligence", with the common purpose of networking the more disparate artefacts found in everyday life: from wearable devices (e.g. hidden in glasses, buttons, pens, wallets etc.) to smart everyday objects (in some cases, made adaptive by using neural networks), up to the proposal of a "gadgetware" architecture style.

E. Mobile and personal communications and satellite systems – take-up measures

The objective of the "take-up" measures is to facilitate the broader application and rapid take up of mobile and personal communications and systems. The work comprises trials that use and evaluate innovative and advanced, yet not fully established, technologies and solutions such as:
- Multimedia interactive, distributive and asymmetric information services over a range of terrestrial networks (terrestrial cellular, cordless and "indoor" radio systems and networks);
- Implementation and validation of new business scenarios where the benefits of satellite communication systems and services can be clearly established;
- Wireless technologies for evolving and scalable systems and networks (including cellular networks, private wireless networks, fixed wireless broadband access systems, wireless local loop systems, cellular interactive systems, and mobile broadband systems).

Three of the projects active in this area (HOME ON AIR, POS.IT and WIRELESSINFO) are exploiting WAP technology, respectively for home control services, fleet management and electronic payment, and agriculture and forest information. The others are addressing different issues:
- TRIADIS will validate the "Distributed Speech Recognition" technique in a real Mobile Network (GSM, DCS-1800), also for possible adaptation to GPRS and UMTS.
- RADIATE will conduct trials to validate their proposal for digital broadcasting in the AM bands around 30MHz (DRM). They expect to show that the expected improvement in audio/data quality and reception reliability are consistently achieved. Also issues of compliance with broadcast legislation and co-existence with analogue broadcasts will be addressed.
- WIN will develop and demonstrate a cellular wireless IP-based communication platform (2-10 Mbit/s, based on Spread Spectrum technology and operating in the 3.6 GHz band), which will cover the networking needs of a medium size area cell (e.g. apartment buildings, businesses, university and school campuses, etc.). The project will design and deploy the wireless network, survey the market and select subscribers' groups, develop user specified and E-commerce applications, as required, and measure post-deployment network performance and subscriber acceptability and satisfaction.
- WITNESS aims to upgrade, test, and validate equipment and planning algorithms to aid the standardisation of a digital terrestrial television (DTT) return channel that can be implemented across Europe, and in other territories adopting the DVB standards. It will upgrade the technology already developed in the earlier project iTTi.

F. Accompanying measures and other related Action Lines

In addition to the previously mentioned research projects, the IST programme allows for funding of "accompanying measures" dealing with activities supporting the implementation of the EU research policy, such as:
- LOCUS, which will help defining the future Emergency Call Service (ECS) in Europe taking into account all aspects of a possible implementation such as: user needs, institutional issues, technical and technological issues, foreseen markets and convergence with other applications. This will provide orientation for the EC policy regarding the most relevant European regulatory framework for enhanced 112 emergency services.
- WSI, whose objective is to focus research capacities on strategic objectives and essential issues through active information and the formation of Innovation Cells, also offering a platform that enables synergy among projects in the wireless area of the IST Programme and disseminating information. The goal is to significantly reduce the time-to-market for future wireless products and services through timely experimentation with research prototypes and collaboration with leading-edge users in trials, also to accelerate the emergence of standards and to support interoperability in and between wireless technologies.

In addition to the focused action lines specific to the mobile communications domain which have been mentioned in the previous paragraphs, the workprogramme 2001 includes some new "cross key action" ALs which strengthen the focus on the development of a pervasive computing environment for seamless access to services. These include actions which:
- Focus on networked embedded information systems and "intelligent" resource-constrained devices in decentralised networks, including in particular the areas of consumer applications and wireless networking in limited mobility environments (action line IV.1.1).
- Focus on the development of middleware (adaptation layer) for seamless access to scalable, personalised and interactive services, over a range of heterogeneous transport networks (terrestrial and satellite, mobile and fixed, wireless and wire based, symmetric and asymmetric, public and private). The work addresses issues of personal mobility (to facilitate location and terminal-

independent working) and service customisation and portability (through various user interface profiles and subscriber profiles). It also covers the application of multi -bearer -streaming techniques, splitting multimedia traffic between bearers of different latency, bandwidth, etc (action line IV.1.2).

Moreover, in order to further stimulate integration between IST Key Actions, there are wider and multidisciplinary "Cross-Programme Actions" (CPA). Some of these have direct links to wireless communications, namely:

- CPA3 "use of geographic information", which also spans the development and demonstration of location based mobile services integrating global positioning and fixed/wireless network technologies, to allow context-based information management and retrieval;
- CPA6 "next generation networks", which promotes large scale experiments of novel IP infrastructures resulting from the convergence of fixed, mobile and wireless technologies and architectures from a service perspective;
- CPA13 "advanced signal processing systems and applications", one of whose foci concerns specifically the application areas of communications (fixed, mobile, wireless, navigation).

In this context it is worth mentioning the project TONIC, resulting from the CPA7 "socio-economic analysis for the Information Society". It aims to evaluate the new business models associated with offering IP based mobile services in a competitive context, also in terms of cost and benefits of providing fixed broadband access to both competitive and non-competitive areas, so to suggest the most appropriate network infrastructure from an economic viewpoint.

Another kind of integration activity are the "cross-programme clusters", launched through accompanying measures (action line VIII.8.1), whose objective is to build links between existing projects to reinforce their complementarity and synergies. On the other side, Networks of Excellence and working groups (action line VIII.8.2) have as objective to bring together a critical mass of industrial and academic research groups to co-ordinate their research or other activities in order to advance towards common strategic goals.

III. THE ARCHITECTURE IN PLACE

In summary, Figure 6 reports a "map" of the IST projects currently running in the mobile communications area. The horizontal axis provides a distinction based on the simplified OSI layer primarily targeted by the research (Air Interface, Network, Application), while the vertical axis defines different groups based on the coverage characteristics of the system considered:

- **Satellite**: global coverage and global mobility through satellite (low bit rate per user, complementing terrestrial segments to provide economically viable global coverage);
- **Terrestrial broadcast**: wide coverage and mobility for broadcast or distribution services (such as DAB-T or DVB-T: long range and big cell size, any other access systems can be used as a return channel);
- **Cellular**: full mobility and roaming for individual links maximising the system capacity. It encompasses evolutions of GSM (2G+: GPRS, EDGE) and 3rd generation systems (3G: IMT-2000/UMTS);
- **WLAN or PAN**: local hot spot coverage for very high data rate asymmetric individual links (e.g. IEEE 802.11, HIPERLAN) or for "universal" wireless networking (e.g. Bluetooth and DECT);
- **Fixed wireless**: providing wireless local loops, e.g. for new operators.

This map shows for example that a large part of the research activity is focused on network aspects of 3G cellular systems, and many projects deal with applications of 2G+

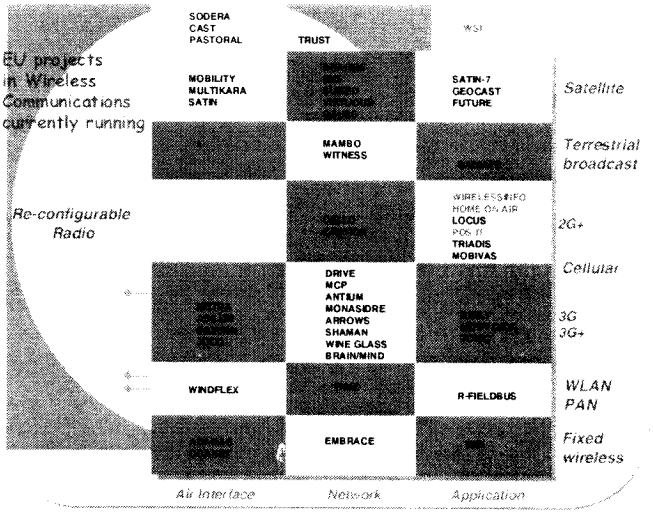


Figure 6 – Map of the projects currently running in the Mobile and Satellite Communications area of IST. The horizontal axis is divided in three portions corresponding to a simplified OSI structure, while the vertical axis distinguishes between different radio coverage characteristics of the systems (as explained in the text). The position of the project name indicates the main area of activity, and the links point to any other areas of interest of the project.

(taking advantage of GPRS and EDGE). One can also easily see that several projects realise a high degree of integration between fixed, mobile, on-line and broadcasting technologies (e.g. DRIVE, MCP, MOBIVAS, MAMBO, SUITED, IBIS, BRAIN, MOBY DICK, just to name a few). This kind of integration is also one of the main objectives specified in the workprogramme for 2001. Other foci of research activities lie in the development and interconnection of embedded technologies, and in the development of middleware, distributed systems and multi-layered architectures to enable inter-operability, interworking, openness and integration of applications and services across platforms. The use of open source software for the application levels is also strongly encouraged.

IV. CONCLUSIONS

This paper aimed at providing an overview of the recent

achievements and current developments of European Union funded IST-R&D programme. Clearly many if not all the issues referred to are of a global nature and require close collaboration and consensus building amongst telecommunications operators, equipment, content and service providers as well as academia and research institutes. Since these issues are very often related to standards and specification work or relate to the future planning of the frequency spectrum, the IST Programme has been opened to participation beyond Europe. Indeed the basic condition for participating in EU-funded R&D actions is that the consortium submitting a research proposal include at least 2 partners from EU countries or one from an EU country and one from an Associated State (i.e. from the EEA, the EU Candidate Member States, Israel and Switzerland). While all the partners from EU or Associated States can be funded, partners from other countries can participate to the activities under self-financing, excepted special cases in which their participation is considered essential to achieve the objectives of the project and therefore Community funding can be envisaged.

Acknowledgements

Special thanks to our colleagues in unit E4 of DG Information Society for their continued support and to the members of the many R&D projects who have been making major contributions to the establishment of a European wireless information society.

ADAMAS	***Broadband fixed wireless access for asymmetric services***
ANTIUM	***Advanced NeTwork radio Identification equipment for Universal Mobile communications***
ARROWS	***Advanced Radio Resource Management for Wireless Services***
ASILUM	***Advanced Signal processing for Link capacity increase in UMTS***
BRAHMS	***Broadband Access for High Speed Multimedia via Satellite***
BRAIN	***Broadband Radio Access for IP based Networks***
CAST	***Configurable radio with Advanced Software Technology***
CAUTION	***CApacity Utilization in cellular networks of present and future generaTION***
CELLO	***Cellular network optimisation based on mobile location***
DRIVE	***Dynamic Radio for IP-Services in Vehicular Environments***
EMBRACE	***Efficient Millimetre Broadband Radio Access for Convergence and Evolution***
EMILY	***European Mobile Integrated Location sYstem***
FUTURE	***Functional UMTS Real Emulator***
GAUSS	***Galileo and UMTS Sinergetic System***
GEOCAST	***Multicast Over Geostationary EHF Satellites***
HOME ON AIR	***WAP Access for a Home Automation Server***
IBIS	***Integrated Broadcast Interaction System***
JOCO	***Joint source-channel Coding-driven digital baseband design for 4G multimedia streaming***
LOCUS	***Location of Cellular Users for Emergency Services***
MAMBO	***Multi-Services Management Wireless Network With Bandwith Optimisation***
MCP	***Multimedia Car Platform***
METRA	***Multi-Element Transmit and Receive Antennas***
MIND	***Mobile IP based Network Developments***
MOBILITY	***Mobile Real Time Tv Via Satellite Systems***
MOBIVAS	***Downloadable MOBIle Value Added Services through Software Radio & Switching Integrated Platforms***
MOBY DICK	***Mobility and Differentiated Services in a Future IP Network***
MONASIDRE	***Management Of Network And Services In A Diversified Radio Environment***
MULTIKARA	***Multibeam Ka-band Receiving antenna for future "multimedia via satellite, direct to home" systems***
OBANET	***Optically Beam-formed Antennas for adaptive broadband fixed and mobile wireless access NETworks***
PASTORAL	***Platform And Software for Terminals: Operationally Re-configurAbLe***
POS.IT	***Terrestrial Wireless Services for Courier Management***
RADIATE	***Radio DIgital Am TEsts***

R-FIELDBUS	***High Performance Wireless Fieldbus In Industrial Related Multi-Media Environment***
SATIN-0	***Satellite-UMTS IP-based Network***
SATIN-7	***An innovative SATellite-based public Internet terminals Network***
SATURN	***Smart Antenna Technology in Universal bRoadband wireless Networks***
SHAMAN	***Security for Heterogeneous Access in Mobile Applications and Networks***
SODERA	***Re-configurable RADIO for Software Defined Radio for 3rd Generation mobile terminals***
SUITED	***Multi-Segment System For Broadband Ubiquitous Access To Internet Services And Demonstrator***
TONIC	***TechnO-ecoNomICs of IP optimised networks and services***
TRIADIS	***Trials on Distributed speech recognition for mobile network embedded voice-controlled services***
TRUST	***Transparent Reconfigurable UbiquitouS Terminal***
VIRTUOUS	***Virtual Home UMTS on Satellite***
WIN	***Wireless Internet Network Objectives***
WIND-FLEX	***Wireless Indoor Flexible High Bitrate Modem Architecture***
WINE	***Wireless Internet Networks***
WINE GLASS	***Wireless IP Network as a Generic Platform for Location Aware Service Support***
WIRELESSINFO	***Wireless supporting of agricultural and forestry information systems***
WITNESS	***Wireless Interactive Terrestrial Network Systems***
WSI	***Wireless Strategic Initiative***

Table 4 Acronyms and full titles of the IST projects mentioned in this paper

Groups of Synchronizers in Digital Communications

António D. Reis[1], José F. Rocha, Atilio S. Gameiro, José P. Carvalho[1]

[1]Dep. de Fisica, Universidade da Beira Interior Covilhã, 6200 Covilhã, Portugal

Resumo - Neste artigo abordamos três importantes grupos de sincronizadores, nomeadamente o de fase, o de símbolo e o de bloco. Contudo dedicamos aqui a nossa atenção aos sincronizadores de símbolo onde distinguimos três grandes classes: a de malha aberta, malha mista e malha fechada.

Os sincronizadores de símbolo têm por objectivo amostrar os dados na máxima abertura do seu diagrama de olho, para retemporizar a duração dos bits com a mínima taxa de erros.

São vários os factores que desoptimizam o ponto de amostragem, mas destacamos o erro de fase estático (desajuste) e o erro de fase aleatório (jitter).

Os sincronizadores de simbolo podem ainda ser analisados pela sua gama de sincronismo e pela sua gama de captura.

Palavras chave: Sincronismo em Comunicações Digitais

Abstract - In this paper we deal with three important groups of synchronizers, namely the phase one, the symbol one and the block one. However here we dedicate our attention to the symbol synchronizers where we distinguish three big classes: the open loop, the mixed loop and the closed loop ones.

The symbol synchronizers have by objective to sample the data in the maximum opening of its eye diagram, to retiming the bits duration with the minimum bit error rate.

There are several factors that lead to a non-optimum sampling point, but we detach the static error phase (misadjust) and the random error phase (jitter).

The symbol synchronizers still can be analyzed by its lock range and capture range.

Key words: Synchronism in Digital Communications

I. INTRODUCTION

In data communication systems the signal must be transmitted and received, in analog systems with the minimum distortion and in digital systems with the minimum bit error rate.

A periodic signal is characterized by its amplitude and frequency (f=1/T), however when this signal is transmitted its amplitude varies with the distance emitter-receiver and its frequency varies with the relative speed emitter-receiver.

Then the analog systems which are sensible to the amplitude must possess an AGC (Automatic Gain Control) circuit that adjusts its gain maintaining an optimized output (ex. 1v) with minimum distortion and the digital systems must possess an AFC (Automatic Frequency Control) circuit that adjusts its frequency maintaining an optimized sampling (ex. $\Delta\phi$=0) with minimum bit error rate and correct retiming.

The effect of the AGC is attached to the linear part (front-end) and the AFC effect is attached to the non linear part (synchronizer), therefore the last one will be analyzed here.

In this work we study the open loop synchronizer (without AFC), the mixed loop synchronizer (with partial AFC) and the closed loop synchronizer (with total AFC).

We will analyze the 3 classes of synchronizers, in terms of static error phase (misadjust) with the input frequency variation (similar to the effect of mistuned components), in terms of synchronism / capture range and still in terms of random error phase (jitter) with the noise (SNR) [1].

Fig.1 shows the block diagram of a general synchronizer.

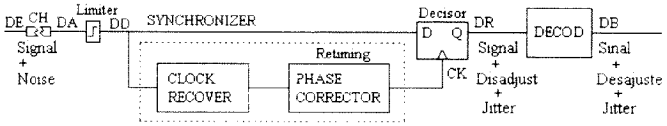


Fig.1 Block diagram of a general synchronizer

Some closed loop synchronizers have the phase corrector and the decisor circuit inlayed in the proper clock recover.

Fig.2 shows qualitatively as the static error phase (misadjusting) and the random error phase (jitter) increases the bit error rate.

The waveforms correspond to the points marked in the circuit. So DE is the data at the emitter, DA the analog data at the receiver input, DD the data with digital format, CK the recovered clock, DR the regenerated data at the synchronizer output and TE the error rate. DB is the data block.

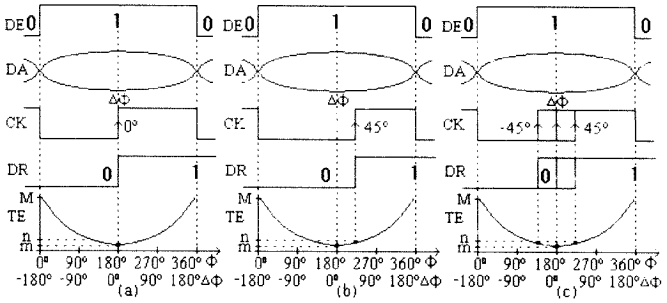


Fig.1 The clock samples the input signal

In Fig.1a there isn't misadjusting nor jitter, then the sampling occurs in the maximum opening of the eye diagram and the error probability is always minimum, in Fig.1b the misadjusting of 45° increases the error rate and in Fig.1c the jitter provokes random increasing of the error rate. If we have present the two error phases (static and random) the total effect aggravates the error probability, which varies with the random error phase $\Delta\phi$ between [0°, 90°].

If the synchronizer is out of synchronism, the error phase varies uncontrolled between [-180°, 180°] and the effect will be catastrophic with the error probability to shoot up.

II. SYMBOL SYNCHRONIZERS

A. Open Loop Synchronizer

The open loop synchronizer will be here represented by the tank circuit but could be any other open loop circuit as the case of the SAW (Surface Acoustic Wave) circuit or the digital circuits (monostable or astable) [2, 3, 4, 5].

Fig.3 represents the circuit tank, which is driven by data impulses at multiple intervals of the period.

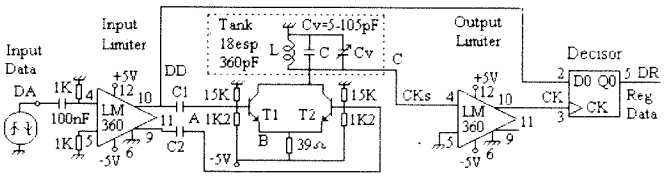


Fig.3 Open loop synchronizer (tank)

The 2 transistors in parallel operate in alternation one on the positive transitions and the other on the negative ones.

Fig.4 illustrates the functioning mode of the open loop synchronizer based in the circuit tank.

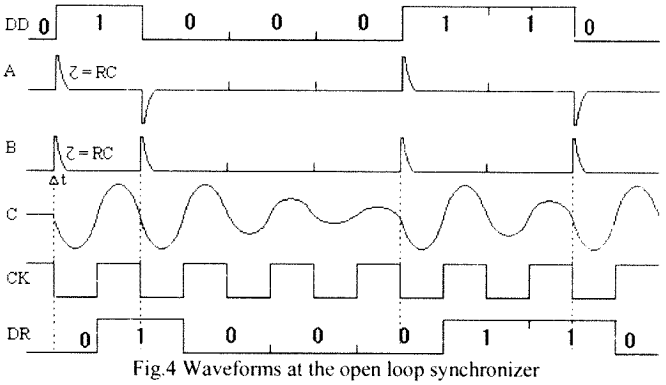


Fig.4 Waveforms at the open loop synchronizer

The input limiter gives digital format to the received signal. The differentiator concentrates the signal energy in the transitions at multiple intervals of the period that carries with itself a strong spectral line that after is selected by the high Q broadband filter. Then the comparator output catches this damped sinusoidal signal and convert it to a rectangular signal which is the recovered clock.

This circuit due to its simplicity allows high transmission rates but however the clock is of limited quality.

B. Mixed Loop Synchronizer

The mixed loop synchronizer shown in Fig.5 uses a conventional PLL after the open loop circuit, with intention to improve the features of the clock [6].

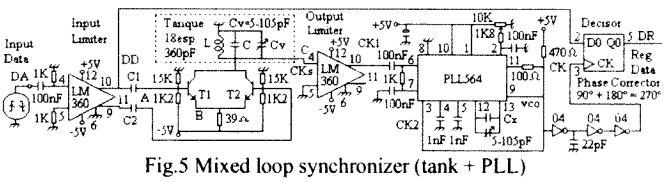


Fig.5 Mixed loop synchronizer (tank + PLL)

The mixed loop synchronizer is evidenced by an open loop circuit followed of a closed loop circuit.

Fig.6 shows the waveforms that illustrate the functioning mode of the mixed loop synchronizer.

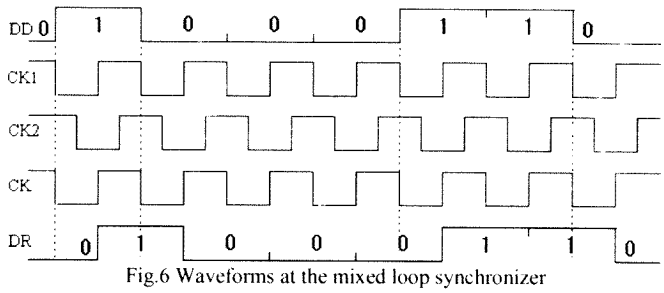


Fig.6 Waveforms at the mixed loop synchronizer

The operation of this circuit can be understood in 2 distinct parts, in the first one an open loop circuit (tank) produces a clock of lesser quality, in the second one a closed loop circuit (PLL) accept this input and provides an output version of good quality.

This synchronizer leaves out of the loop a considerable part of the total circuit, because only gets to synchronize its VCO with a signal which has a similar deterministic harmonic.

C. Closed Loop Synchronizer

The closed loop synchronizer can synchronize directly its VCO with the input random data and then all the components are inside of the loop [7].

Fig.7 shows the closed loop synchronizer where the VCO triggers the flip flop that samples the input data.

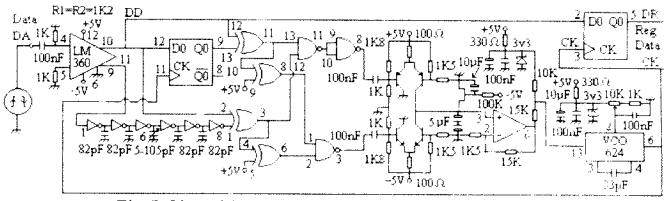


Fig.7 Closed loop synchronizer (triggered by VCO)

Fig.8 shows the waveforms that illustrate the functioning mode of the closed loop synchronizer.

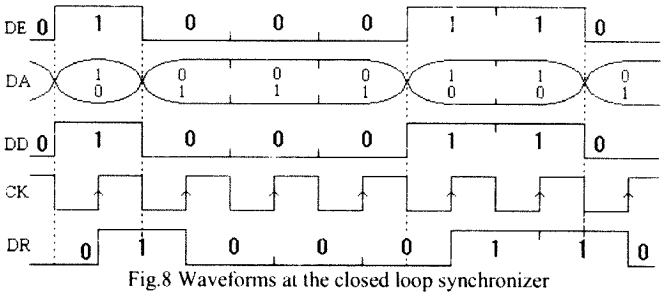


Fig.8 Waveforms at the closed loop synchronizer

The operation of this circuit is based on the comparison of a variable pulse against another fixed of reference (T/2).

When a data transition occurs initiates a variable pulse that finishes in the next positive transition of clock so that, the sampling (positive transition of the clock) occurs in the bit center. The duration of the variable pulse must be also equal

consecutive transitions of the clock.

III. DESIGN, TESTS AND RESULTS

A. Test setup

To study the 3 mentioned synchronizers, we established three different comparisons, the first one at level of static error phase (misadjusting), the second one at level of synchronism / capture range and the third one at level of random error phase (jitter).

Fig.9 shows the general test setup, which allow to obtain the experimental and simulation results [10].

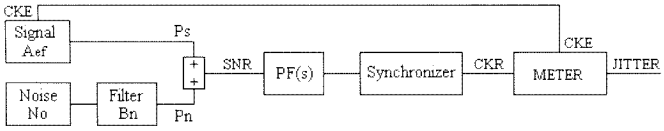


Fig.9 Block diagram of the test setup

The two first comparisons was made experimentally using the oscilloscope as the real measure device. The third comparison was made experimentally with the measure device (ANDO), but some difficulties in controlling the noise in a "GAP", obliged us later to resort to the simulation.

For this reason, we created the equivalent simulation models of the real circuits and after we used a powerful mathematical tool to make the processing.

The design and dimensioning of the circuits were made by form to provide equal conditions and can be meet in [8].

The prefilter was not used in this work (PF(s)=1), but can be useful for high noise quantities [9].

B. Results at level of static error phase

Fig.10 shows the experimental results obtained with an oscilloscope, for the 3 classes of synchronizers, when the input varies its transmission rate (frequency).

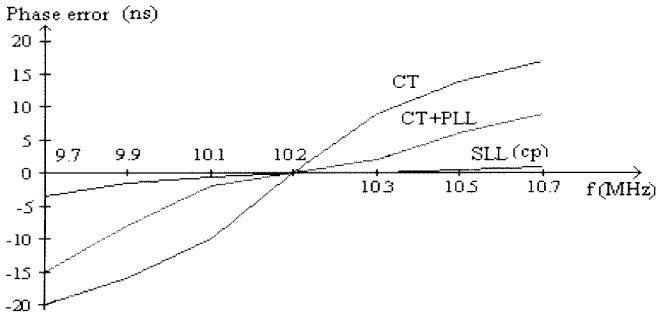


Fig.10 Static error phase (misadjusted)

Identical results would be waited if we maintain the input, but on the other hand we mistuned the circuits.

We verify that the open loop synchronizer is the one that produces greater static error phase for the same variation of the input frequency. After, the mixed loop synchronizer begins to feel the effect of a partial closed loop diminishing the error phase. Finally, the closed loop synchronizer possess a total loop between the VCO and input, therefore any misadjusting is strongly reduced, by the loop, to a low value.

We must still point out, that the closed loop synchronizer can increment the loop gain, reducing the static error, however must be preserved the loop stability.

C. Results at level of synchronism / capture range

Fig.11 shows the results of the 3 classes of synchronizers at level of operation / synchronism range.

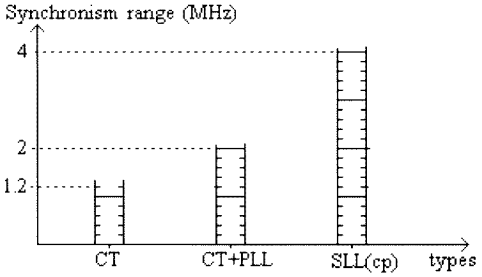


Fig.11 Operation / synchronism range

We verify that the closed loop synchronizer possess greater operation/synchronism range than the mixed one and this has a bigger range than the open loop one.

Fig.12 shows the results of the 3 classes of synchronizers at level of operation / capture range.

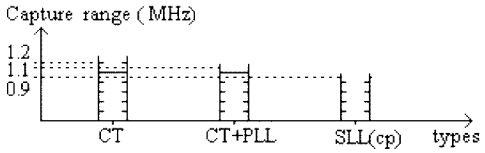


Fig.12 Operation / capture range

We verify that the closed loop synchronizer possess minor operation/capture range than the mixed one and this a lesser range than the open loop one.

D. Results at level of random error phase

Fig.13 shows the output jitter in UIRMS (Unit Intervals Root Mean Squared) as function of the input SNR (signal to noise relation) for the 3 cited synchronizers.

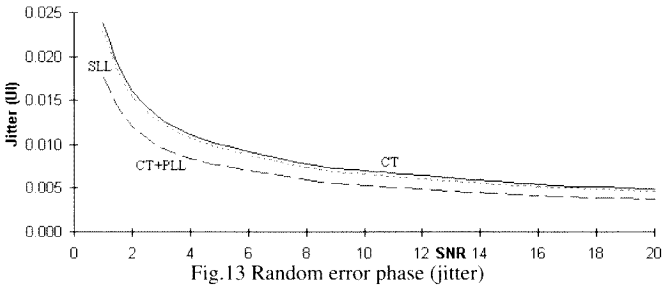


Fig.13 Random error phase (jitter)

We verify that the open and closed loop synchronizers have similar jitter performance but the mixed one has a slightly advantage due to its double tuning, especially for low SNR. For high SNR the 3 synchronizers tend to be identical.

These results were measured with parameters that provide identical linear operation modes for the 3 synchronizers.

The open and mixed loop synchronizers do not possess AFC total and this imposes that the filter broadband Bf be sufficiently large to avoid the risk to go out of tuning when the input varies.

On the other hand, the closed loop synchronizer possess AFC total and scanning circuit, then the VCO (extracted clock) nearly follows the input transmission rate, so a narrowband filter can be used what implies a reduced loop noise bandwidth Bl and consequently a lower jitter.

IV. ANOTHER CLOSED LOOP SYNCHRONIZER

A. Synchronizer triggered by input

The closed loop synchronizer that we shown previously was triggered by VCO (VCO acting the flip-flop clock) now we go to present another synchronizer but that is triggered by input (entered acting the flip-flop clock). Both closed loop synchronizers are sequential because they possess memory.

In Fig.14, the proposed synchronizer possesses auxiliary scanning circuit, which strongly increases the capture range to a value similar to the synchronism range.

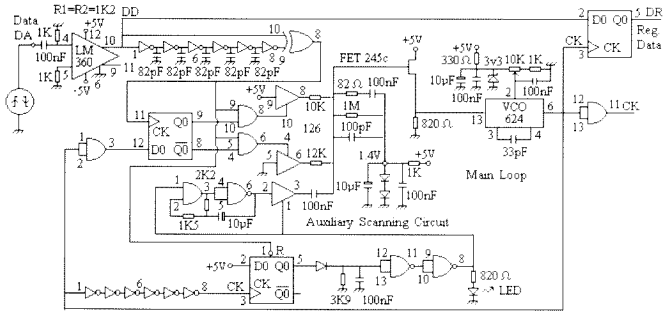


Fig.14 Closed loop synchronizer triggered by input

Tab.1 shows the effect of the scanning circuit

Tab.1 – Synchronism and capture ranges

SYNCHRONISM RANGE Transmission rate (frequency MHz)	Phase error (ns)
11	+ 1
10.6	+ 0.25
10.2	+ 0+
10	0
9.8	− 0+
9.4	− 0.5
9	− 1.5
8.5	− 2.5
8	− 5

8 --- 10 --- 11
1M + 2M = 3 MHz
(a)

CAPTURE RANGE Transmission rate (frequency MHz)	Phase error (ns)
10.2	+ 0+
10	0
9.8	− 0+

9.7 --- 10.2 --- 10.8
200 + 200 = 400 KHz
− --> Delayed clock
+ --> Advanced clock
(b)

CAPTURE RANGE WITH AID Transmission rate (frequency MHz)	Phase error (ns)
11	+ 1
10.6	+ 0.25
10.2	+ 0+
10	0
9.8	− 0+
9.4	− 0.5
9	− 1.5
8.5	− 2.5
8	− 5

8 --- 10 --- 11
1M + 2M = 3 MHz
(c)

We verify that the capture range without aid is incremented, becoming similar to the synchronism one.

By this way the only possible disadvantage of the closed loop synchronizer is also favorably solved.

When the synchronism is restored, the synchronism detector, with luminous signaling, disactivates the scanning generator, leaving it in high impedance, for not disturbing the functionality of the main circuit.

B. Jitter-noise curves of the 2 closed synchronizers

Fig.15 shows the experimental curves of jitter-SNR (optical power variation) for the 2 closed loop synchronizers.

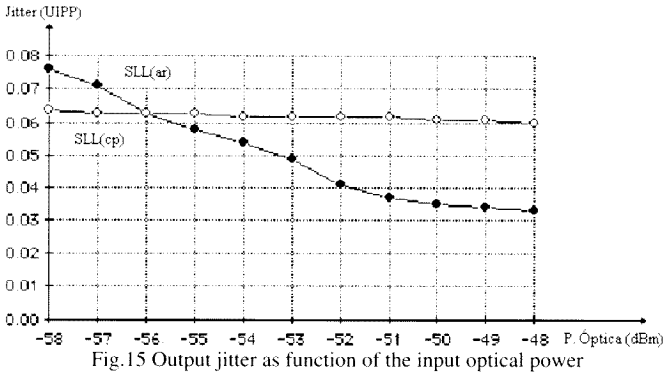


Fig.15 Output jitter as function of the input optical power

We verify that for low SNR (Ps<-56dBm) the triggered one by VCO gains advantage, but for high SNR (Ps>-56dBm) the triggered one by input presents some advantage.

Fig.16 shows the simulation curves of output jitter UIRMS as function of the input SNR for the 2 synchronizers.

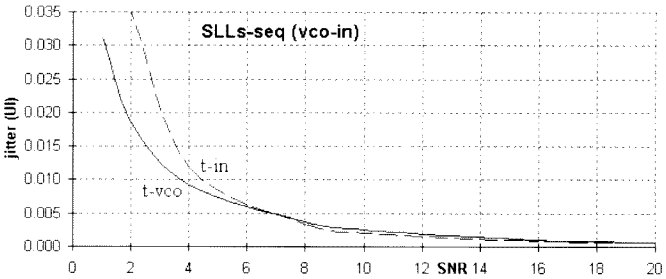


Fig.16 Output jitter UIRMS as function of the input SNR

We verify that for low signal noise relations (SNR<6) the triggered by VCO gains advantage, but for high signal to noise relation (SNR>6) is the triggered one by input that presents a slightly advantage.

The optical signal power Ps and the noise signal power Pn are related by the formula (SNR=Ps/Pn).

Generally we can say, such as in the experimental jitter-noise curves, also in the simulation ones we verified that for low SNR the synchronizer triggered by VCO gains some advantage, but for high SNR is the synchronizer triggered by input that gains a slightly advantage.

C. Output jitter versus input frequency jitter

We made other more experimental tests over the 2 closed loop synchronizers. The test now consisted to measure the output jitter amplitude, when it is injected an input signal with constant jitter amplitude (0.2 UIPP (Unit Interval Peak to Peak)) but with variable frequency.

Fig.17 shows the curves output jitter - input jitter when the entered keeps its amplitude (0.2 UIPP) but varies the frequency.

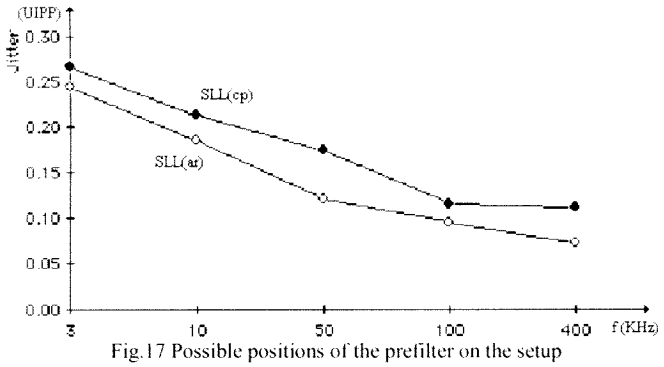


Fig.17 Possible positions of the prefilter on the setup

We verify that the output jitter amplitude diminishes, when the input jitter frequency increases, although one maintain its amplitude (0.2 UIPP).

V. CONCLUSIONS

We considered 3 classes of symbol synchronizers namely the open loop, the mixed loop and the closed loop, then we established comparisons at level of static error phase (misadjusting), at level of synchronism / capture range and at level of random error phase (jitter).

We verify that the closed loop synchronizer only can present some disadvantage in relation to the capture range but that can also be recouped with the aid of the slow scanning circuit.

The results show that the closed loop synchronizer guarantees the lesser static error phase (misadjusting) and consequently also the greater stabilization of its components. This synchronizer endowed with scanning circuit also possesses the best operation/synchronism/capture range.

For the same bandwidth of the filter Bf / loop noise bandwidth Bl, the 3 classes of synchronizers present similar curves of random error phase (jitter). However it must be mentioned that the closed loop one possess AFC and therefore the loop filter can be reduced without problems, with the consequent reduction of the loop noise bandwidth and logically also with the reduction of the jitter.

VI. ACKNOWLEDGMENTS

The first author, Antonio D. Reis is thankful to the program PRODEP for financial support.

VII. REFERENCES

[1] – Werner Rosenkranz, "Phase Locked Loops with limiter phase detectors in the presence of noise", IEEE Transactions on Communications com-30. October 1982.

[2] - A. H. Jazwinski, "Filtering for Nonlinear Dynamical Systems" IEEE Automat. Contr. p.765 October 1966.

[3] - M. B. N. Butler, "SAW Circuit", Electronic Engineering p.37, July 1980.

[4] - B. A. Galwas, "Scattering Matrix Description of Microwave Resonators", IEEE Trans. Microwave Theory and Technics p.669 August 1983.

[5] - H. H. Witte, "A Simple Clock Extraction Circuit Using a Self Sustaining Monostable Multivibrator Output Signal". Electronic Letters, October 1983.

[6] - J. C. Imbeaux, "performance of the delay-line multiplier circuit for clock and carrier synchronization", IEEE, J. Sel. Areas in Com., Jan. 1983.

[7] - Charles R. Hogge, "A Self Correcting Clock Recovery Circuit", Journal of Lightwave Technology, Dec. 1985.

[8] – A. D. Reis, J. F. Rocha, A. S. Gameiro, J. P. Carvalho "Bit Synchronizer Operating at Half Rate", II Conference on Telecommunications p.550, Sesimbra-PT 15-16 April 1999.

[9] – A. D. Reis, J. F. Rocha, A. S. Gameiro, J. P. Carvalho, "Effects of Prefiltering on the Symbol Synchronizer Performance", Revista do DET p.90, Univ. Aveiro, September 2000.

[10] – A. D. Reis, J. F. Rocha, A. S. Gameiro, J. P. Carvalho "A New Technique to Measure the Jitter", III Conference on Telecommunications p.64, F. Foz-PT 23-24 April 2001.

Transmissão de Sinais Ópticos em Banda Lateral Única, com Igualação no Domínio Eléctrico Utilizando o Critério dos Zeros Forçados, para Sistemas com um Ritmo de Transmissão de 10 Gbit/s

Edgardo Costa, António Pinho, Paulo Monteiro, Rui Ribeiro, J. Ferreira da Rocha, Tiago Maia

Resumo – **É apresentado um sistema de transmissão óptico em banda lateral única (SSB) a 10 Gbit/s, que juntamente com o critério de igualação dos zeros forçados, permite a compensação da dispersão no domínio eléctrico. A transmissão de sinais ópticos SSB permite a duplicação das distâncias de transmissão. É demonstrado por simulação do sistema, que para um valor de BER de 10^{-9}, utilizando o método de igualação anterior, se conseguem atingir distâncias de transmissão até 215 km.**

Abstract - **In this article is investigated an optical single sideband transmission system at 10 Gbit/s with electrical equalization, using the zero forcing criteria, to compensate the chromatic dispersion. The optical transmission of single sideband signals increases the transmission distances relative to the transmission in double sideband (DSB) form. Simulation results also showed that using this electrical equalization it is possible to achieve transmission distances up to 215 km, for a BER of 10^{-9}.**

I. INTRODUÇÃO

A dispersão cromática na fibra é um fenómeno que limita significativamente as distâncias de transmissão em sistemas com elevado ritmo de transmissão, como é o caso de sistemas em que a transmissão se efectua a 10 Gbit/s. Isto deve-se ao facto da dispersão cromática provocar um aumento da interferência entre símbolos (IES) no sinal detectado. Torna-se então importante o desenvolvimento de técnicas que permitam compensar os efeitos da dispersão cromática na fibra.

Neste contexto, assumem primordial importância os recentes métodos de transmissão de sinais ópticos em banda lateral única (SSB-*"Single Sideband"*) para sistemas em banda base a operar a 10 Gbit/s [1].

A transmissão de sinais ópticos em banda lateral única apresenta vantagens em relação à transmissão em banda lateral dupla (DSB-*"Double Sideband"*). Para o caso de um sinal no formato SSB, a largura de banda óptica é aproximadamente metade em comparação com um sinal DSB, reduzindo-se assim significativamente os efeitos indesejáveis da dispersão cromática e possibilitando o aumento considerável do número de canais de transmissão. A detecção directa do sinal SSB presença da portadora, resulta na detecção auto-homodina. O sinal eléctrico detectado apresenta neste caso preservação da informação de fase, o que permite a sua igualação no domínio eléctrico.

A igualação da dispersão pode ser efectuada por uma linha microfita [1], que apresenta uma resposta em frequência aproximadamente inversa da resposta em frequência da fibra. No entanto, para distâncias de fibra consideráveis este tipo de igualação é bastante limitativo, uma vez que existe um comprimento de linha microfita óptimo para cada comprimento de fibra. Para além disso, para grandes comprimentos de fibra podem ser necessárias linhas microfita com tamanhos consideráveis, o que provoca uma atenuação significativa e dependente da frequência, causando distorção do sinal.

Neste artigo é apresentada uma técnica que consiste na criação de sinais ópticos em banda lateral única (SSB), que juntamente com uma posterior igualação eléctrica do sinal detectado, através do método dos zeros forçados, permite a compensação da dispersão introduzida pela fibra.

Na secção II é descrita em detalhe a teoria associada á criação de sinais ópticos de banda lateral única (SSB) e a teoria da igualação pelo método dos zeros forçados. Na secção III são apresentados e discutidos os resultados obtidos por simulação de um sistema óptico a 10 Gbit/s. São apresentados os resultados relativos à transmissão de sinais SSB, os resultados da igualação eléctrica do sinal através do critério dos zeros forçados e é feita uma comparação entre este tipo de igualação e a igualação teórica. Por fim, na secção IV são apresentadas as conclusões finais do trabalho realizado.

II. TEORIA

A. *Transmissão de sinais em banda lateral única*

No método utilizado para a criação de sinais ópticos em banda lateral única (SSB), a informação do sinal e a sua transformada de Hilbert são aplicadas a um modulador Mach-Zehnder (MZ), onde as portadoras ópticas sofrem um desfasamento relativo de $\pi/2$ radianos no ponto onde são adicionadas.

Para criar um sinal óptico SSB segundo o método descrito anteriormente, utiliza-se a configuração representada na figura 1. A configuração utiliza um modulador Mach-Zehnder (MZ) e um modulador de fase (PM).

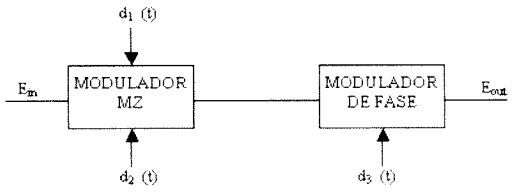


Fig. 1 - Configuração utilizada para a geração de sinais ópticos em banda lateral única (SSB)

O campo eléctrico à saída do modulador MZ pode ser representado pela expressão (1), onde V_π representa, a tensão de inversão total de fase no modulador, $d_1(t)$ e $d_2(t)$ representam os sinais eléctricos de modulação de cada ramo do modulador e $Ein(t)$ representa o campo eléctrico à entrada do modulador, que é dado por $exp(j\omega_0 t)$, onde ω_0 representa a frequência da portadora óptica.

$$Eout(t)=\frac{Ein(t)}{2}\left[exp\left(j\pi\frac{d_1(t)}{V\pi}\right)+exp\left(j\pi\frac{d_2(t)}{V\pi}\right)\right] \quad (1)$$

O campo eléctrico à saída do modulador de fase é dado pela expressão (2), onde $d_3(t)$ representa o sinal eléctrico de modulação do PM.

$$Eout(t)=\frac{Eout_{MZ}(t)}{2}\left[exp\left(j\pi\frac{d_3(t)}{V_\pi}\right)\right] \quad (2)$$

Os sinais eléctricos de modulação $d_1(t)$, $d_2(t)$ e $d_3(t)$, são dados pelas expressões (3), (4) e (5) respectivamente, onde x representa o factor de modulação, $m(t)$ é uma versão do sinal de dados (NRZ) onde se removeu a componente contínua e que apresenta uma gama de valores compreendida entre –0.5V e 0.5V. O sinal $\hat{m}(t)$ é a transformada de Hilbert de $m(t)$.

$$d_1(t)=xV_\pi m(t)-\frac{V_\pi}{4} \quad (3)$$

$$d_2(t)=-xV_\pi m(t)+\frac{V_\pi}{4} \quad (4)$$

$$d_3(t)=xV_\pi \hat{m}(t) \quad (5)$$

A implementação experimental de uma aproximação da transformada de Hilbert, por forma a ser criado o sinal $\hat{m}(t)$ a partir do sinal de dados $m(t)$, é feita utilizando o filtro transversal (filtro FIR) com quatro atrasos apresentado em [1].

B. Igualação utilizando o critério dos zeros forçados

Com a igualação utilizando o critério dos zeros forçados ("*Zero-Forcing Equalizer*") [2], pretendemos determinar o filtro com função de transferência $H_r(f)$ que para impulsos $g(t)$ na entrada origine impulsos $h_0(t)$ na saída com interferência entre símbolos (IES) nula, e que para esta condição minimize a variância do ruído nos instantes de amostragem. Para tal, é necessário que as caudas precursoras e posteriores do impulso elementar à saída do filtro $H_r(f)$ se anulem nos instantes afastados de múltiplos de T do instante de amostragem. Se considerarmos o instante de amostragem em t=0, o filtro igualador $H_r(f)$ terá de verificar as seguintes condições:

$$\begin{cases} h_0(0)=1 \text{ (ou outra constante diferente de 0)} \\ h_0(KT)=0, \quad K=-N,\ldots,N \wedge K\neq 0 \end{cases} \quad (6)$$

O filtro óptimo $H_r(f)$ que garante as condições descritas em (6) é dado pela seguinte expressão.

$$H_r(f)=\frac{G^*(f)}{S_n(f)}\cdot\sum_{k=-N}^{N}\lambda_k\, e^{-j.2\pi.k.T} \quad (7)$$

Na expressão anterior $S_n(f)$ representa a densidade espectral de potência (DEP) do impulso $g(t)$ recebido. O igualador não é mais que a cascata de um filtro adaptado com função de transferência $G^*(f)/S_n(f)$, com um filtro transversal (filtro FIR) com 2N+1 coeficientes e 2N atrasos. O filtro adaptado tem por objectivo maximizar a relação sinal ruído no instante de decisão. Este filtro não anula a IES e pode mesmo aumentá-la, por isso utiliza-se um filtro FIR de comprimento 2N+1 para remover a IES causada por 2N símbolos (N símbolos anteriores e N símbolos posteriores). O número de coeficientes do filtro é igual ao número de condições que se pretende impor.

O valor dos coeficientes λ_k do filtro transversal são calculados resolvendo o sistema de equações lineares que em termos matriciais é dado por:

$$L^T=U^T\cdot M^{-1} \quad (8)$$

$$L=\begin{bmatrix}\lambda_{-N}\\ \vdots\\ \lambda_0\\ \vdots\\ \lambda_N\end{bmatrix} \text{ e } U=\begin{bmatrix}0\\ \vdots\\ 1\\ \vdots\\ 0\end{bmatrix}$$

$$M=\begin{bmatrix} m(0) & m(T) & \cdots & m(2N-1)T) & m(2NT) \\ m(-T) & m(0) & \cdots & m(2N-2)T) & m(2N-1)T) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m(-(2N-1)T) & m(-(2N-2)T) & \cdots & m(0) & m(T) \\ m(-2NT) & m(-(2N-1)T) & \cdots & m(-T) & m(0) \end{bmatrix}$$

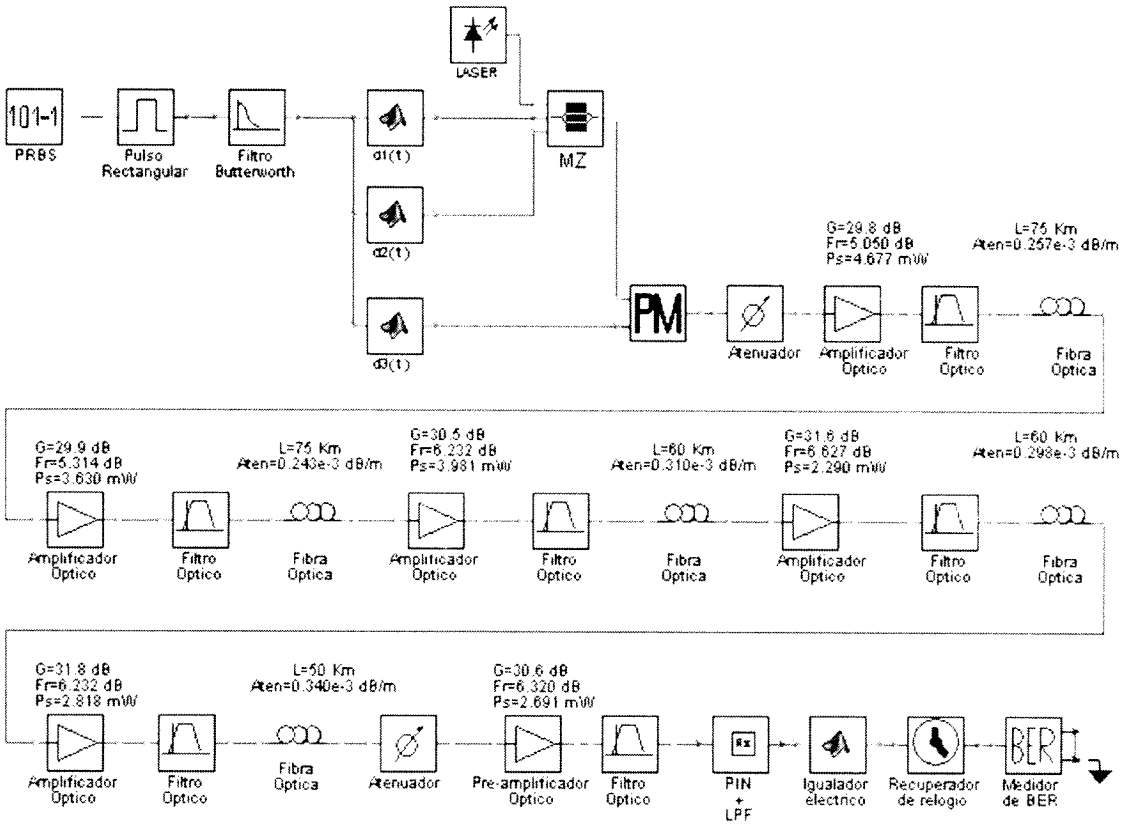


Fig. 2 – Esquema utilizado na simulação do sistema óptico a 10 Gbit/s.

III. RESULTADOS DAS SIMULAÇÕES

Para se proceder ao teste por simulação do sistema de transmissão óptico em banda lateral única a 10 Gbit/s, com igualação da dispersão no domínio eléctrico utilizando o critério dos zeros forçados, utilizou-se o esquema representado na figura 2. O sistema óptico foi implementado no simulador VPItransmissionMaker^{TM} [3].

O emissor SSB é composto por um gerador de sequências pseudo-aleatórias, seguido de um formatador de impulsos eléctricos, para se efectuar a criação do sinal de dados *m(t)* com formato NRZ. Seguidamente faz-se passar o sinal por um filtro passa-baixo Butterworth de 3ª ordem com frequência de corte a 3 dB igual a 6.5 GHz. Usa-se um laser como fonte óptica, com uma frequência central de 193.55 THz, o que equivale a 1550 nm em termos de comprimento de onda e com uma largura espectral de 150 KHz e amplitude 1 mW. A portadora óptica é modulada pelo sinal de informação a transmitir através de um modulador de amplitude MZ e posteriormente por um modulador de fase PM. O sinal à saída do modulador de fase PM vai passar por um atenuador óptico para se conseguir igualar a potência óptica média do sinal ao valor referido em [1]. O sistema óptico tem 320 km de fibra e é constituído por 5 troços de fibra óptica com amplificação, de forma a podermos compensar as perdas da fibra. Após cada amplificador introduziu-se um filtro óptico passa-banda que tem como objectivo filtrar parte do ruído de emissão espontânea introduzido pelos amplificadores. Trata-se de um filtro passa-banda trapezoidal com frequência central igual a 193.55 THz, largura de banda de passagem igual a 20 GHz, largura de banda entre a frequência de corte superior e a frequência de corte inferior de 40 GHz e atenuação de 20 dB quer a uma quer a outra frequência. As características dos amplificadores utilizados, no que diz respeito ao factor de ruído, ganho e potência de saturação são apresentadas na figura 2, junto ao respectivo amplificador. A fibra utilizada é denominada de NLS (*Nonlinear Dispersive Fiber*) e apresenta uma dispersão (D_λ) de $17*10^{-6}$ s/m^2, dispersão de 2ª ordem ($S_\lambda = \partial D_\lambda / \partial \lambda$) de $0.08*10^{-3}$ s/m^3, índice de refracção não linear de $2{,}6*10^{-20}$ m^2/W e considerou-se nulo o efeito de Raman. A atenuação de cada troço de fibra bem como o seu comprimento encontram-se representadas mais uma vez na figura 2. Por fim, o receptor utilizado no sistema, consiste num fotodiodo PIN (*Positive-Intrinsic-Negative*) e num filtro passa-baixo de Bessel de 3ª ordem com frequência de corte a 3 dB igual a 6.5 GHz, que tem por objectivo simular a largura de banda limitada de um PIN experimental. O fotodiodo PIN utilizado é caracterizado por uma responsividade R=0.7
A/W, corrente de escuridão de 1 nA, ruído térmico de $12*10^{-12}\ A/\sqrt{Hz}$, tendo-se também incluído ruído quântico gerado na detecção óptica do sinal.

A. *Transmissão de sinais em banda lateral única*

Com o auxilio de um analisador de espectros óptico, colocado à saída do modulador de fase obteve-se o espectro óptico do sinal SSB gerado, que é apresentado na figura 3 a). Mostra-se também na figura 3 b), o espectro óptico de um sinal DSB. Para a obtenção do sinal DSB procedeu-se à alteração da saída do bloco responsável pela criação do sinal eléctrico de modulação de fase $d_3(t)$, de forma a que este sinal fosse nulo. Desta forma o modulador de fase fica inoperante e o sinal modulado à saída do modulador MZ é o sinal DSB pretendido. É de referir que os resultados foram obtidos para um factor de modulação x=0.2 para o caso do sinal SSB e para um factor de modulação x=0.5 no caso do sinal DSB. Para os dois casos o valor de V_π utilizado é de 1 V. O valor do atraso T do filtro de Hilbert, foi considerado como sendo igual a 37.5 ps.

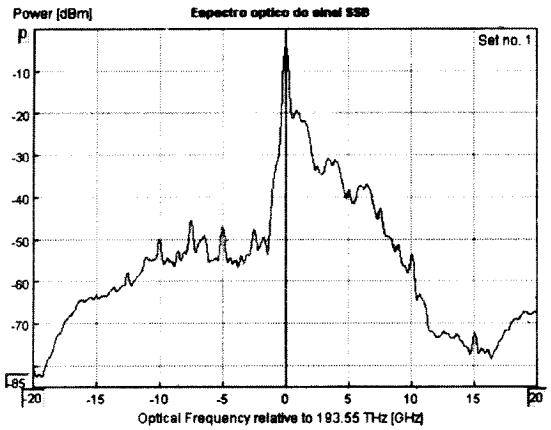


Fig. 3 – Espectro óptico (a) do sinal SSB e (b) do sinal DSB para um sistema a 10 Gbit/s.

Pela análise da figura 3 a) conclui-se que existe uma redução de cerca de 30 dB do espectro, na gama de frequências inferiores à frequência central.

Na figura 4 está representada a sensibilidade do receptor em função do comprimento da fibra óptica, para os casos do sinal DSB e do sinal SSB sem igualação. Nos dois casos o valor da sensibilidade foi medido de modo a garantir-se um valor de BER ("*Bit Error Rate*")de 10^{-9}.

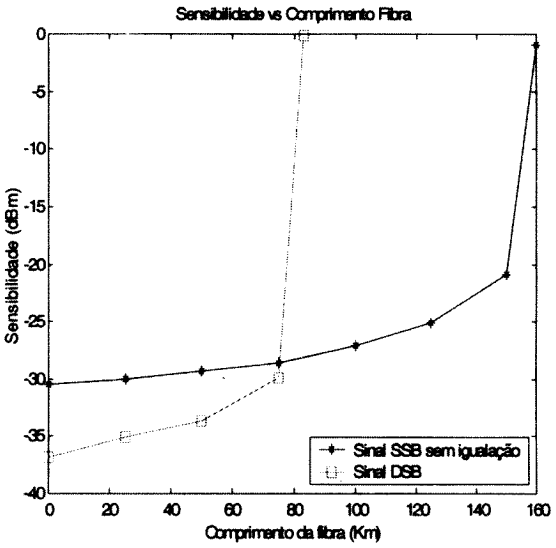
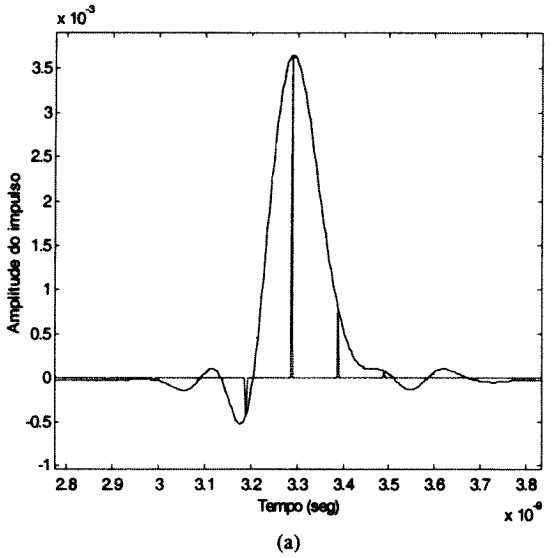


Fig. 4 – Sensibilidade do receptor em função do comprimento da fibra, para uma BER de 10^{-9}, para o caso do sinal DSB e do sinal SSB sem igualação, para um sistema a 10 Gbit/s.

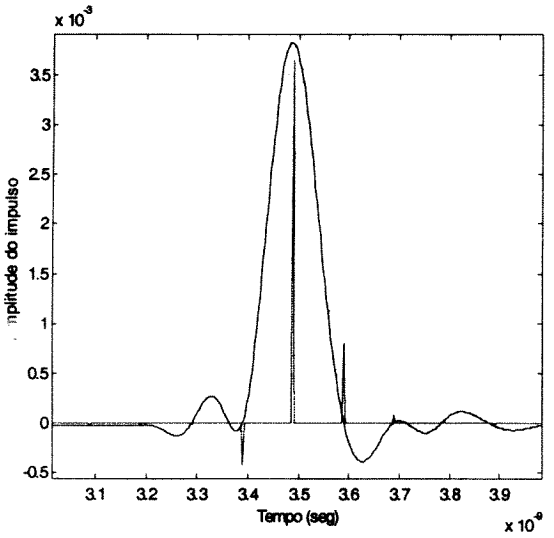
Para comprimentos de fibra relativamente pequenos, os valores da sensibilidade para o caso do sinal DSB são superiores aos valores da sensibilidade para o caso do sinal SSB sem igualação. Isto deve-se ao facto do factor de modulação ser superior no caso do sinal DSB, o que provoca um aumento da potência média à saída do modulador Mach-Zehnder. Garantindo um valor de BER de 10^{-9} consegue-se transmitir eficientemente o sinal DSB ao longo de 80 km de fibra e o sinal SSB sem igualação ao longo de 160 km de fibra. Conclui-se, que com este método de transmissão de sinais ópticos em banda lateral única, as distâncias de transmissão são aumentadas para o dobro. Isto deve-se ao facto do sinal SSB apresentar uma largura de banda óptica que é aproximadamente metade da largura de banda óptica do sinal DSB.

B. *Igualação utilizando o critério dos zeros forçados*

Para se proceder à igualação no domínio eléctrico do sinal SSB utiliza-se o critério dos zeros forçados. Para tal, torna-se necessário calcular os coeficientes do filtro transversal de 4ª ordem (5 coeficientes) que é utilizado. Pretende-se o anulamento do impulso nos dois instantes de amostragem anteriores (-T e –2T) e nos dois instantes de amostragem posteriores (+T e +2T), em relação ao instante de amostragem de referência para o qual o impulso apresenta o seu valor máximo. O filtro possui atrasos de 1 período de bit T=100 ps. Na figura 5 a) e 5 b) estão representados o impulso inicial sem igualação e o impulso final com igualação obtido à saída do filtro transversal igualador, para 150 km de fibra.



(a)



(b)

Fig. 5 – Impulso (a) inicial, antes da igualação e (b) final, após a igualação, para 150 km de fibra.

O impulso anula-se nos dois instantes de amostragem anteriores e posteriores ao instante de amostragem para o qual o impulso apresenta o seu valor máximo. Para comprimentos de fibra superiores verifica-se um aumento da distorção do impulso e um alargamento do mesmo, o que provoca um aumento da amplitude das caudas precursoras e posteriores do impulso final igualado.

Tendo em consideração que os coeficientes do filtro transversal igualador foram calculados para diversos comprimentos de fibra procedeu-se ao estudo do desempenho do igualador ZF. Na figura 6 está representada a sensibilidade do receptor em função do comprimento da fibra óptica, para os casos do sinal DSB, do sinal SSB sem igualação e do sinal SSB com igualação utilizando o filtro transversal de 4ª ordem.

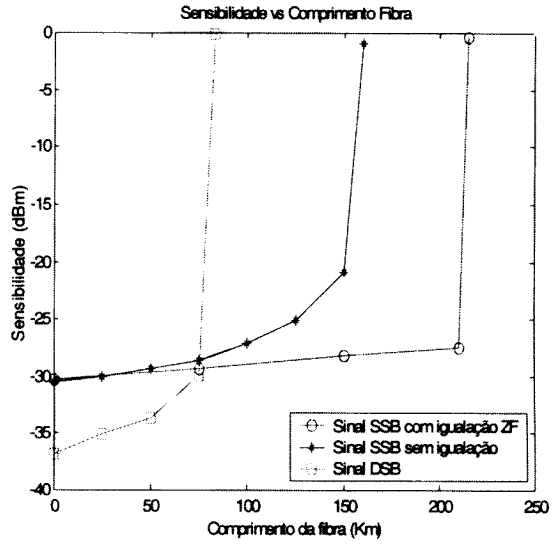
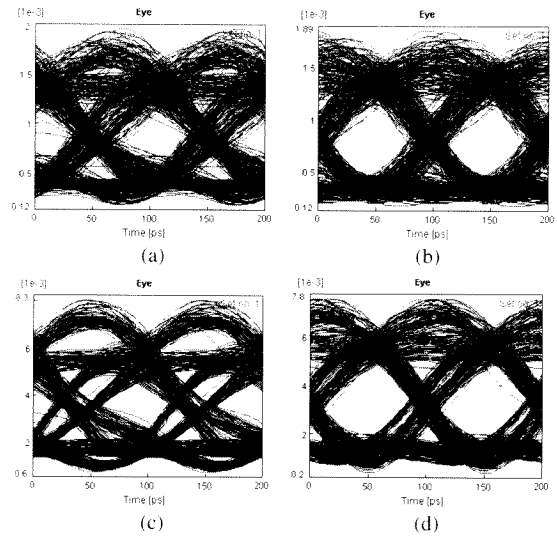


Fig. 6 – Sensibilidade do receptor em função do comprimento da fibra, para uma BER de 10^{-9}, para o caso do sinal DSB, do sinal SSB sem igualação e do sinal SSB com igualação pelo método dos zeros forçados, para um sistema a 10 Gbit/s.

Com este método de igualação conseguem-se distâncias de transmissão de 215 km para um valor de BER de 10^{-9}. Se compararmos esta distância com a obtida sem igualação do sinal SSB, verifica-se que se consegue um acréscimo de cerca de 55 km. Na figura 7 representam-se os diagramas de olho do sinal SSB antes e depois da igualação, para 150 km e 210 km de fibra. Verifica-se que para estas distâncias de fibra a utilização do igualador é fundamental, conseguindo-se aumentar consideravelmente a abertura do olho.



(a) (b) (c) (d)

Fig. 7 – Diagramas de olho do sinal SSB (a) antes da igualação e (b) depois da igualação para 150 km de fibra e (c) antes da igualação e (d) depois da igualação para 210 km de fibra.

C. *Igualação teórica versus igualação pelo critério dos zeros forçados*

Nesta secção realiza-se um estudo comparativo entre a igualação teórica e a igualação utilizando o critério dos zeros forçados.

Fazendo a aproximação para pequenos sinais, a igualação eléctrica da dispersão em termos teóricos, pode ser efectuada utilizando um modelo equivalente dado pela função de transferência representada pela expressão (9) [1,4], onde D é o coeficiente de dispersão, L é o comprimento da fibra, λ é o comprimento de onda de trabalho, f é a frequência equivalente passa-baixo e c a velocidade da luz.

$$\begin{cases} H(f) = \exp\left(-\dfrac{j\pi DL\lambda^2 f^2}{c}\right), & f \geq 0 \\ H(f) = \exp\left(\dfrac{j\pi DL\lambda^2 f^2}{c}\right), & f < 0 \end{cases} \quad (9)$$

Na figura 8 representa-se a sensibilidade do receptor em função do comprimento da fibra, para o caso da igualação teórica utilizando o filtro com função de transferência representada pela expressão (9). Apresentam-se também os valores da sensibilidade para o caso da igualação com filtro transversal de 4ª ordem utilizando o critério dos zeros forçados e para o caso em que não se efectua qualquer igualação do sinal.

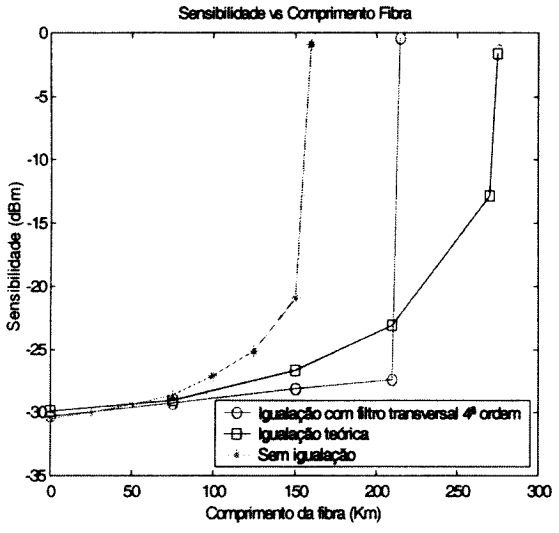


Fig. 8 Sensibilidade do receptor em função do comprimento da fibra, para uma BER de 10^{-9}, do sinal SSB sem igualação, do sinal SSB com igualação pelo método dos zeros forçados e do sinal SSB com igualação teórica, para um sistema a 10 Gbit/s.

Utilizando o igualador teórico, garante-se a transmissão com um valor de BER igual a 10^{-9} até cerca de 275 km. Isto representa um acréscimo de cerca de 60 km comparativamente ao caso em que se utiliza o filtro transversal de 4ª ordem. Conclui-se que o filtro transversal igualador de 4ª ordem consegue uma melhoria significativa no desempenho do sistema. No entanto, não se conseguem atingir os resultados obtidos com o filtro teórico.

V. CONCLUSÕES

Verifica-se que o sinal em banda lateral única (SSB) consegue ser razoavelmente aproximado pelo filtro transversal FIR com 4 atrasos. Apesar da banda lateral não ser completamente eliminada, consegue-se uma redução de cerca de 30 dB do espectro das frequências à esquerda da frequência central, o que é bastante significativo.

No que diz respeito ao desempenho do emissor SSB, verifica-se que as distâncias de transmissão aumentam aproximadamente para o dobro, em comparação com o caso da emissão de sinais DSB. O aumento para o dobro da distância de transmissão conseguida com o sinal SSB, deve-se ao facto do sinal SSB apresentar uma largura de banda óptica que é aproximadamente metade da largura de banda óptica do sinal DSB.

Com a igualação utilizando o critério dos zeros forçados, conseguem-se distâncias de transmissão de 215 km para um valor de BER de 10^{-9}. Se compararmos esta distância com a obtida sem igualação do sinal SSB, verifica-se que se consegue um acréscimo de cerca de 55 km.

O filtro transversal igualador de 4ª ordem apresenta resultados muito melhores que o caso em que não se efectua igualação do sinal SSB. No entanto, não se conseguem atingir as distâncias de transmissão alcançadas com o igualador teórico.

Foi demonstrado que a utilização de um filtro transversal de 4ª ordem, para a realização da igualação pelo método dos zeros forçados, é uma boa escolha para a igualação do sinal em banda lateral única no domínio eléctrico.

REFERÊNCIAS

[1] Mike Seiben, Jan Conradi and David E. Dodds, "Optical Single Sideband Transmission at 10 Gb/s Using Only Electrical Dispersion Compensation", *J. of Lightwave Technology*, vol. 17, No 10, pp 1742-1749, Outubro 1999.

[2] Atílio Gameiro, Apontamentos da Disciplina de Processamento de Sinal em Comunicações Digitais.

[3] Manuais do Simulador VPItransmissionMaker™

[4] A. F. Elrefaie, R. E. Wagner, D. A. Atlas, and D. G. Daut, "Chromatic Dispersion Limitations in Coherent Lightwave Transmission Systems", *J. of Lightwave Technology*, vol. 6, pp. 704-709, May 1988.

Optimization of an Optical Single Sideband Transmitter

Tiago Maia, Rui Ribeiro, Paulo Monteiro

Resumo - **O crescente interesse na modulação de sinais em banda lateral única para transmissão digital de informação em fibras ópticas, deve-se à melhor eficiência do espectro do sinal e à redução da distorção produzida pela dispersão cromática da fibra.**

Neste artigo é analisado o impacto da profundidade de modulação, e é estudada a implementação do filtro de Hilbert no emissor, com o intuito de optimizar o desempenho de um sistema óptico. Para o sistema óptico a 10 Gbit/s estudado, concluiu-se que o valor óptimo obtido para a profundidade de modulação é de 0.2, e que um filtro de Hilbert com 4 baixadas e um atraso temporal elementar de 37.5ps, mostra-se adequado.

Abstract – **The interest in single sideband modulation for transmission of digital data along optical fibers is growing, since it allows an improvement on the signal spectral efficiency and a reduction on the distortion produced by the fiber chromatic dispersion.**

In the optical single sideband transmitter the influence of the Hilbert filter configuration and the modulation depth were analysed in order to optimize the performance of the optical system. For the 10 Gbit/s optical system presented, an optimum value of 0.2 is obtained for the modulation depth and a 4-tap transversal filter structure with an elementary time delay of 37.5ps is shown to be an appropriate solution to implement the Hilbert transform.

I. INTRODUCTION

In order to reduce the effects of the chromatic dispersion the transmission of optical single sideband signals has received great interest in the last few years.

The bandwidth of optical single sideband (OSSB) signals is approximately one half of the corresponding conventional double sideband signals. This bandwidth reduction improves the signal tolerance to the fiber chromatic dispersion and permits a higher channel density in wavelength division multiplexing (WDM) networks. Furthermore, if the OSSB signal is combined with the corresponding carrier, then its optical direct detection can be taken as a self-homodyne detection. This technique preserves the phase information in the detected signal and therefore permits linear dispersion equalization in the electrical domain.

The difficulty with most of SSB transmission techniques [1] is that they require filter designs, which are matched to the frequency separations of interest and are therefore limited in bandwidth [2,3,4]. A new approach is proposed by [5] in which is possible to generate optical SSB signals in digital band or in sub carrier's applications. Moreover in self-homodyne OSSB systems that make use of this technique the modulation depth of the transmitted signal is a crucial parameter. The influence of the modulation depth for a 10 Gbit/s system with different transmission lengths is investigated based in simulation experiments, and the optimum value is found.

In Section II, it's described the complete optical system link studied, in particular the optical source technique for SSB transmission. In Section III the simulation results demonstrate the influence of the modulation depth and the filter time delay on the performance of an optical system at 10 Gbit/s. The concluding remarks are given in Section IV.

II. SYSTEM MODEL

The system that was carried out through the simulation is presented in figure 1. In this system, a master optical carrier at 1550 nm is generated by a laser diode. The OSSB modulation is based on the Hartley concept [6], where combinations of the information signal and its Hilbert transform modulate two quadrature optical carriers. The modulation circuit consists in a dual arm Mach-Zehnder (MZ) modulator followed by a phase modulator. The electrical waveforms required to drive the two modulators, in order to generate a chirp-free OSSB signal, are described in [5,7]. If this optical signal is attached to a photodiode, which corresponds to square-law detection, then the electrical signal at the detector output will be given by (1), where the Taylor series has been used to expand the signal till the third order. In (1), $m(t)$ represents the binary information sequence, a pseudo-random sequence at 10 Gbit/s in our experiments, with levels of +0.5 and -0.5 for the two binary digits, and $\hat{m}(t)$ is its Hilbert transform; V_π is the modulator's switching

voltage. The magnitude of the signal modulation can be strained by the parameter x, which hereafter will be designed by *modulation depth*.

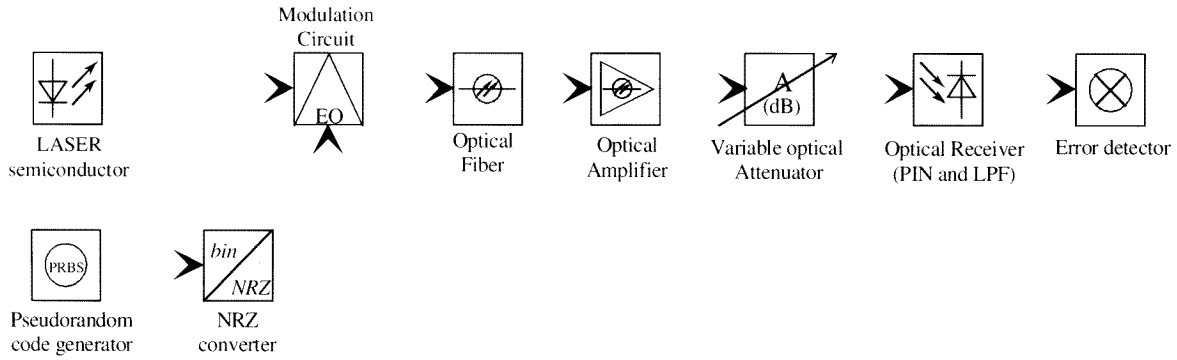


Fig.1 - Optical SSB system configuration.

$$\left|Eout(t)\right|^2 = \frac{1}{2} + xV_\pi m(t) - \frac{2}{3}(xV_\pi)^3 m^3(t) + \frac{1}{4}(xV_\pi)^3 m(t)\hat{m}^2(t) + \ldots \quad (1)$$

From (1) it can be seen that the information signal $m(t)$ is present in the detected electrical signal, but there are other terms that are potential sources of interference. In the absence of fiber chromatic dispersion, the second order powers of the signal cancel out, as it can be seen in (1), but this is no more observed in the presence of chromatic dispersion. Since the magnitude of the undesirable terms in the detected signal is proportional to higher powers of the modulation depth x, the value for this parameter should be kept small.

An approximation of the data Hilbert transform, which is required by the phase modulator in the SSB signal generation, is obtained by filtering the original information signal with a tapped delay filter [5]. The discrete approximation of the Hilbert filter is shown in figure 2. The cell delay to produce the Hilbert transform is in fact 2T. The central tap added to the circuit allows to append the carrier to the modulated signal. To obtain only the Hilbert transform, the α parameter in figure 2 should be null.

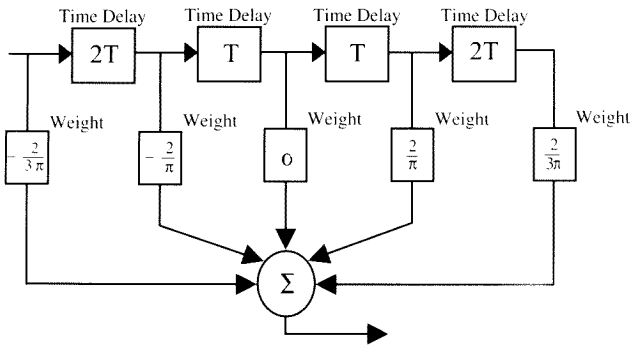


Fig. 2 – 4-tap Hilbert filter.

Figure 3 shows the power spectrum of the OSSB signal for a modulation depth of 0.2 and a filter time delay (T) of 37.5 ps, generated by our system model. In the same figure is show the spectrum obtained when an ideal Hilbert transform filter is used. In these spectra, the suppression of the lower side band is clear, as well as the presence of the optical carrier.

In our system model, a single mode optical fiber has been attached to the optical transmitter, followed by an optical amplifier that compensates the fiber attenuation. The transmitted signal is then detected by a photodiode and processed by a lowpass filter. A linear electrical equalizer pursues the lowpass filtering, which is based on a small–signal analysis and gives rise to a transfer function that inverts the effect of the fiber chromatic dispersion. An error rate estimator, based on a gaussian approximation for the signal distribution, completes the system. The receiver sensitivity implies a bit error rate of 10^{-9}. The simulation tests of the optical SSB transmission system where carried out through a simulator known by *Photonics Transmission Design Suite* (PTDS).

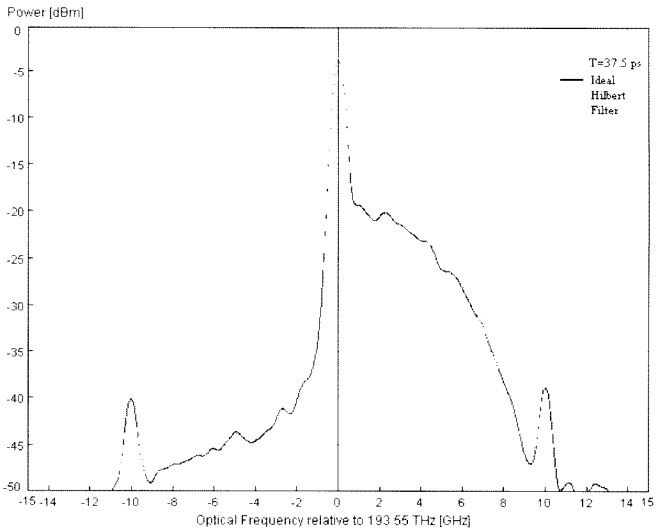


Fig. 3 – Simulated optical spectrum for SSB signals at 10 Gb/s (Gaussian Bandwidth resolution: 0.5 GHz).

III. TRANSMITTER OPTIMIZATION

The optical direct detection of the signal in the presence of the carrier results, actually, in self-homodyne detection. At the receiver, the term of interest will be the beat signal

between the carrier and the SSB signal. Other components, of higher order, appear in the detected signal, which corresponds to signal distortion and thus undesirable. As discussed above, a small value for the modulation depth permits us to limit the power within these unwanted components. However, it also implies a larger fraction of the available power attributed to the carrier, in opposition to the information signal component. Nevertheless, the carrier presence is always necessary in order to obtain self-homodyne detection. Therefore, the optimum value for the modulation depth x, is achieved by a trade-off between distortion and effective signal power.

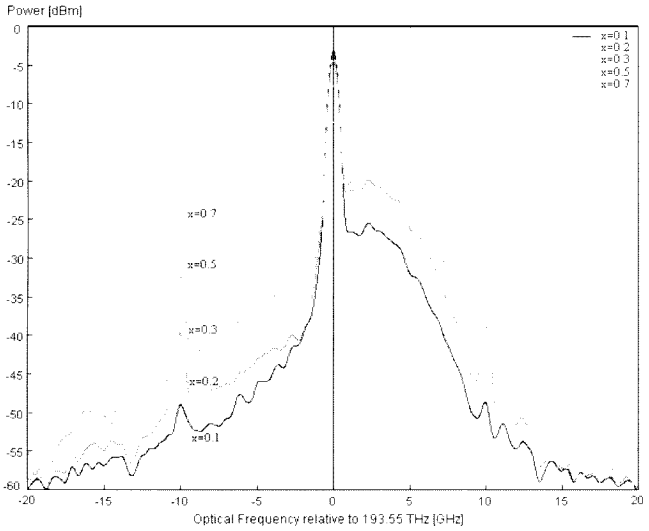


Fig. 4 – Transmitter output power spectrum for different modulation depth values (Gaussian Bandwidth resolution: 0.5 GHz).

The power spectrum at the transmitter output is plotted in figure 4, for several modulation depths. In all the curves, it can be observed the strong attenuation suffered by the frequency components below the center frequency, characteristic of an upper single sideband signal. The optical carrier presence is also evident.

The receiver sensitivity versus modulation depth for several system configurations is shown in figure 5.

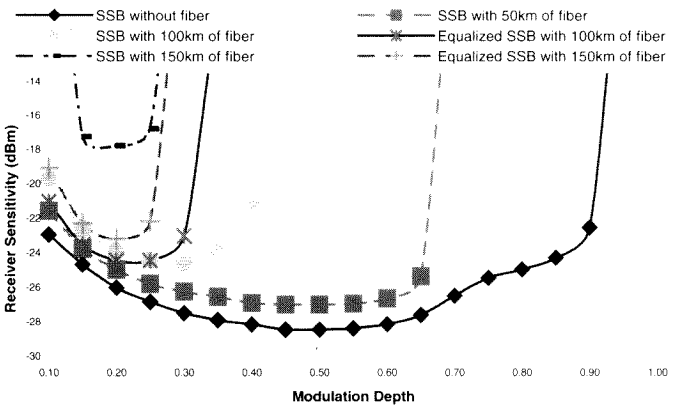


Fig. 5 – Simulated receiver sensitivity versus modulation depth in a 10 Gb/s optical system with a BER of 10^{-9}.

Without optical fiber, the optimum modulation depth is 0.5. This value corresponds to an intensity-modulated signal with an extinction ratio of 100% at the MZ modulator output. As the modulation depth deviates from that value the higher will be relative power attributed to the carrier, which does not contain any information. When 50 km of fiber are considered, without dispersion equalization, the system sensitivity is close to the preceding case for small values of the modulation depth, and the optimum value for this parameter still is 0.5. However, now the sensitivity decays abruptly when the modulation depth exceeds 0.6. The sensitivity degradation is attributed to the presence of the second order terms in the detected signal, which do not cancel out as in the case without fiber. The distortion due to chromatic dispersion and the optical amplifier noise represent other negative contributions to the receiver sensitivity.

The harmful effect of chromatic dispersion is more pronounced for larger fiber lengths. Figure 5, shows the cases of systems with 100 and 150 km of fiber without dispersion equalization. In addition to the observable degradation in the system sensitivity, the optimum value for the modulation depth is also significantly reduced, to a value of about 0.2 in both cases. Figure 5 also presents results for equalized systems with the two above fiber lengths. For these systems, the optimum modulation depth value is again in the vicinity of 0.2. As it can be seen in figure 5, the dispersion equalization is efficient for small values of the modulation depth. For higher values, other phenomena dominate the distortion and the small-signal-based equalization becomes useless. In summary, 0.2 is a suitable value for the modulation depth in OSSB systems regardless of using or not electrical equalization.

The optimization of the transversal filter that implements the Hilbert transform was also performed. A number of 4-tap shows to be a good compromise between circuit complexity and efficiency. Although additional taps do improve the filter amplitude response, the corresponding impact on the optical system performance is negligible.

The influence of the time delay used in the 4-tap Hilbert filter was as well investigated. The receiver sensitivity versus fiber length for different tap delays of the filter is plotted in figure 6. These experiments demonstrate that

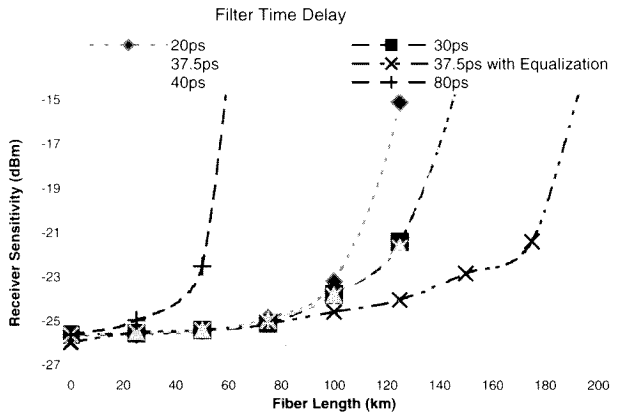


Fig. 6 – Simulated receiver sensitivity versus fiber length at a BER of 10^{-9}.

time delay, particularly for fiber lengths below 100 km. Nevertheless, the optimum value found for that parameter was 37.5 ps.

IV. CONCLUSIONS

Simulation experiments are used to assess the feasibility of OSSB systems using self-homodyne detection. The experiments revealed that the electrical waveforms of the drive signals, required by the transmitter modulators, play a critical role on the system performance. Particularly important is the modulation depth. The present study recommends a modulation depth of 0.2 in order to obtain the best performance. The number of cells and the tap delay used in the discrete approximation of the Hilbert filter were also optimized, resulting in a 4-tap with a time delay of 37.5 ps.

ACKNOWLEDGEMENTS

The work performed by Edgardo Costa e Antonio Pinho is hereby acknowledged.

The authors would also like to thank the Portuguese foundation for Science and Technology (FCT) for their financial support.

REFERENCES

[1] Lee T. Nichols and R. D. Esman, "Single sideband modulation techniques and applications", OFC/IOOC 1999.

[2] J. Parj, W. V. Sorin, and K.Ylau, "Elimination of the fiber chromatic dispersion penalty on 1550nm millimeter-wave optical transmission", Electronic Letters, vol. 33, pp 512-513, 1997.

[3] Kazushige Yonenaga and Noboru Takachio, "A Fiber Chromatic Dispersion Compensation Technique with an Optical SSB Transmission in Optical Homodyne Detection Systems", IEEE Photonics Technology Letters, vol. 5, No 8, pp 949-951, August 1993.

[4] R. Hofstetter, H. Schmuck and R. Heidemann, "Dispersion effects in optical millimeter-wave systems using self-heterodyne method for transport and generation", IEEE Trans. Microwave Theory Tech., vol.43, pp 2262-2269, 1995.

[5] Mike Seiben, Jan Conradi and David E. Dodds, "Optical Single Sideband Transmission at 10Gb/s using Only Electrical Dispersion Compensation", J. of Lightwave Technology, vol. 17, No 10, pp 1742-1749, October 1999.

[6] Bob Davies, Jan Conradi "Hybrid Modulator Structures for Subcarrier and Harmonic Subcarrier Optical Single Sideband", IEEE Photonics Technology Letters, vol. 10, No 4, pp 600-602, April 1998.

[7] Edgardo Costa, Antonio Pinho, Paulo Monteiro, Rui Ribeiro, J. Fereira da Rocha and Tiago Maia, "Transmissão de sinais ópticos em banda lateral única, com igualação no domínio eléctrico utilizando o crtério dos zeros forçados, para sistemas a 10 Gbit/s", submitted to the present issue in this journal.

Utilização de hardware reconfigurável para acelerar a *satisfação booleana*[*]

Iouliia Skliarova, António B. Ferrari

Resumo **– Este artigo apresenta um estudo de possibilidade de aceleração da *satisfação booleana* com a ajuda do hardware reconfigurável. A *satisfação booleana* (SAT) é um problema importante que tem muitas aplicações em CAD e outras áreas. Neste artigo propomos uma técnica de desenvolvimento orientada a problema em geral para acelerar a resolução de SAT formulado sobre matriz discreta. O algoritmo utilizado requer uma unidade de controlo bastante complexa que é implementada inteiramente em hardware reconfigurável. Por fim, são analisadas diferentes possibilidades de resolução de SAT e argumenta-se que os melhores resultados podem ser obtidos com a ajuda da colaboração de uma aplicação de software executada num computador de uso geral com um circuito de solução de SAT implementado em FPGA.**

Abstract **– The paper presents a case study of accelerating Boolean satisfiability in reconfigurable hardware. Boolean satisfiability (SAT) is an important problem having many applications in CAD and other areas. We propose an application-specific approach to accelerate the backtrack search algorithm for the SAT problem formulated over discrete matrix. The algorithm employed involves a quite sophisticated control unit, which is entirely implemented in reconfigurable hardware. Finally, we analyze different possibilities of solving the SAT problem and argue that the best results can be achieved by the use of software, running on a general-purpose computer, together with an FPGA-based reconfigurable SAT solver.**

I. INTRODUCÃO

A computação reconfigurável é uma área bastante nova em que o hardware é programado de acordo com as necessidades de cada aplicação específica. A computação reconfigurável recorre ao uso de componentes lógicos programáveis, tais como FPGAs. Ao contrário dos computadores convencionais que são programados a nível de instruções, os dispositivos reconfiguráveis são programados a nível de componentes funcionais. Isto permite implementar funcionalidades bastante mais ricas e, consequentemente, atingir para certas aplicações um desempenho muito mais elevado que o dos computadores de uso geral. O desempenho elevado é atingido devido aos seguintes factores [1]:

- Largura de banda elevada no acesso à memória;
- Exploração de paralelismo e de *pipelining*;
- Uso de unidades funcionais específicas e optimizadas.

Analisemos estas técnicas em mais detalhe. Tradicionalmente os computadores de uso geral organizam a memória em forma de um conjunto de palavras de tamanho fixo. Os dados de um problema não cabem exactamente numa só palavra, portanto são precisos vários acessos à memoria para os processar. Contudo, em FPGA a organização da memória pode ser feita de acordo com a dimensão dos dados de um problema, portanto estes podem ser processados numa única operação.

Muitas aplicações involvem operações bastante simples mas estas não são bem suportadas por ALUs convencionais. Sendo assim é possível implementar em FPGA a unidade funcional optimizada para certas operações e tamanhos de dados. Esta unidade é normalmente simples e ocupa poucos recursos de hardware portanto é possível reproduzi-la para explorar o paralelismo.

As aplicações que são normalmente implementadas em sistemas reconfiguráveis são as dominadas pelo processamento de dados caracterizadas por estruturas de controlo relativamente simples. Neste artigo propomos a utilização de computação reconfigurável em problemas com estruturas de controlo complexas que surgem em particular na área de optimização combinatória. Para tal escolhemos o problema de *satisfação booleana* que é de grande importância em áreas diversas.

Recentemente foram propostas várias implementações em FPGAs de circuitos de solução de SAT [2, 3, 4]. Todas estas propostas beseiam-se na ideia de geração de um circuito especializado para cada instância do problema a resolver (*instance-specific approach*). A maior vantagem desta estratégia é que o mapeamento directo da função booleana nos componentes funcionais permite incrementar significativamente o desempenho e assugurar uma boa utilização de recursos do hardware. Neste caso o tempo necessário para resolver um problema é composto pela soma do tempo de geração do

[*] Trabalho financiado com a bolsa da FCT-PRAXIS XXI/BD/21353/99

circuito respectivo, tempo de configuração da FPGA e tempo de execução. Existem vários compiladores especiais que aceleram a geração de configurações de FPGA. Contudo, o tempo de compilação do hardware continua a ser bastante elevado e, como regra, é maior que o tempo da execução do algoritmo. Portanto, este método só pode ser usado para problemas com grandes volumes de dados de entrada, quando o tempo de compilação do hardware é amortizado pelo tempo de execução.

A outra questão que afecta cada algoritmo implementado em hardware reconfigurável é a capacidade da plataforma respectiva. Caso o circuito não possa ser implementado numa só FPGA, é possível ocupar vários dispositivos aplicando os métodos especiais de partição entre FPGAs. Contudo não é garantido que um problema arbitrário possa ser resolvido com os recursos de hardware reconfigurável disponíveis.

Tendo em atenção estas questões propomos aplicar a estratégia orientada à aplicação e não à instância do problema. Neste caso o circuito é desenvolvido uma só vez, optimizado e testado. Portanto, o tempo de compilação do hardware pode ser considerado igual a zero, e o tempo de resolução de um problema só é composto por tempo de configuração da FPGA e o tempo de execução. Uma vez que os recursos de FPGA disponíveis são sempre limitados, propomos utilizar uma arquitectura que é baseada na colaboração do software e hardware reconfigurável.

O resto deste artigo está organizado de maneira seguinte. Na secção 2 especifica-se formalmente o problema de *satisfação booleana* e descreve-se o algoritmo utilizado para a sua resolução. Na secção 3 é proposta a arquitecura do circuito que implementa o algoritmo considerado. A seguir, na secção 4, é descrito o modelo de colaboração de software e do hardware reconfigurável. Finalmente, são apresentados os resultados e a sua comparação com os do GRASP [5] – o mais eficiente algoritmo para SAT implementado em software. As conclusões estão na secção 6.

II. PROBLEMA DE SAT E O ALGORITMO UTILIZADO

É conhecido que CNF (*Conjunctive Normal Form*) é a conjunção de um número de cláusulas onde cada cláusula é a disjunção de uma ou mais variáveis ou das suas negações. O problema de *satisfação booleana* (SAT) consiste em determinar se a função em CNF é satisfazível, i.e. se existe tal atribuição de valores às variáveis que faz com que a função tome valor 1. Obviamente, para que a função inteira avalie a 1 é necessário que cada cláusula seja igual a 1. Por exemplo, a função seguinte contem 3 variáveis e 4 cláusulas e é satisfeita quando $x_1=x_2=x_3=1$:

$$(x_1 \vee \overline{x}_3)(\overline{x}_2 \vee x_3)(x_1)(\overline{x}_1 \vee x_2)$$

De outro lado a função seguinte é insatisfazível:

$$(x_1 \vee \overline{x}_3)(x_3)(\overline{x}_1)(\overline{x}_1 \vee x_2)$$

O problema de SAT pode ser especificado sobre vários modelos matemáticos tais como funções booleanas e matrizes discretas. Um modelo pode ser formalmente transformado noutro. Escolhemos a representação em matrizes porque estas são de bastante fácil processamento em FPGA.

Formulemos o problema de SAT sobre a matriz **M**. Para isso vamos fazer corresponder a cada cláusula c_i uma linha da matriz m_i e a cada variável x_j uma coluna da matriz m_j. Se a variável x_j entra na cláusula c_i então o elemento respectivo da matriz m_{ij} é igual a '1', se a variável x_j entra na cláusula c_i com a negação então o elemento respectivo da matriz m_{ij} é igual a '0', e se a variável x_j não entra na cláusula c_i então o elemento respectivo da matriz m_{ij} é igual a '-' (*don't care*). Por exemplo, a função seguinte:

$$(x_1 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee x_2) \qquad (1)$$

pode ser representada com a matriz **M**:

$$\mathbf{M} = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 \end{matrix} \\ \begin{bmatrix} 1 & - & 0 \\ 0 & 1 & - \\ - & 0 & 0 \\ 1 & 1 & - \end{bmatrix} \end{array} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix}$$

É de notar que resolver um problema de SAT é equivalente ao encontrar um vector ternário **v** que seja ortogonal a cada linha da matrix **M** respectiva. Caso seja impossível arranjar tal vector então a função correspondente é insatisfazível. Por outro lado caso encontremos o vector **v**, devemos invertê-lo. Os zeros e uns no vector invertido apontarão àqueles variáveis que devem ser iguais a 0 e 1 respectivamente para satisfazer a função. Para a matriz **M** apresentada em cima a solução é o vector **v** = -01. Sendo assim, $\overline{\mathbf{v}}$ = -10, i.e. $x_2=1$ e $x_3=0$. É fácil verificar que atribuição destes valores às variáveis satisfaz a função (1).

Para encontrar o vector ortogonal a cada linha de uma matrix ternária aplicámos o algoritmo proposto em [6] apresentando-o em forma de uma árvore de pesquisa [7]. Neste caso a cada vértice da árvore corresponde um vector ternário **v** e uma matriz **M**′ composta por vários menores da matriz **M**. Inicialmente o vector **v** está totalmente indeterminado, i.e. **v**="-...-", e **M**′=**M**. A transição de um vértice da árvore de pesquisa a outro efectua-se se reduzirmos a matriz **M** ou atribuirmos valor 0 ou 1 a uma componente do vector **v**. Na fig.1 está apresentado o esquema de blocos do algoritmo utilizado. O esquema mostra que durante a pesquisa aplicam-se vários métodos de redução, depois, quando a redução se torna impossível, efectua-se a decomposição da situação corrente (o que corresponde à ramificação da árvore). A seguir, aplicam-se outra vez os métodos de redução e o algoritmo continua até que encontre a solução ou chegue à conclusão que esta não existe. Os métodos de redução envolvidos no algoritmo são os seguintes:

- Método 1. Na matriz **M**´ apagam-se todas as colunas que são totalmente indeterminadas, i.e. não contêm zeros nem uns.
- Método 2. Na matriz **M**´ apagam-se todas as linhas que são ortogonais ao vector **v** corrente.
- Método 3. Na matriz **M**´ apagam-se todas as colunas que correspondem às componentes determinadas do vector **v**.
- Método 4. Caso na matriz **M**´ exista uma linha que tem uma só componente com o valor 0 ou 1 e todas as outras suas componentes iguais a '-', então à componente correspondente do vector **v** atribui-se o valor inverso.
- Método 5. Caso na matriz **M**´ exista uma coluna que não contem valor 0 (ou 1), então este valor é atribuido à componente correspondente do vector **v**.

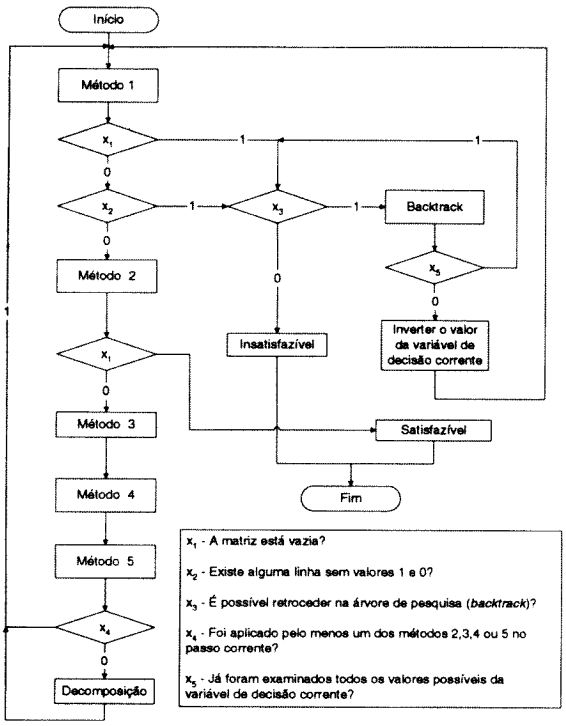


Fig. 1 – O algoritmo de resolução de SAT formulado sobre uma matriz discreta

Na decomposição de uma situação efectua-se o exame sequencial de valores 1 e 0 de alguma componente do vector **v**. Deve ser escolhida a componente que corresponda à coluna mais determinada da matriz **M**´, i.e. à coluna que tenha o número mínimo de valores '-'. Sendo assim efectua-se a *selecção dinâmica* da variável de decisão seguinte. Para cada variável de decisão o valor 1 é examinado antes do valor 0.

Caso depois de apagar uma linha da matriz **M**´ esta se torne vazia então o valor corrente do vector **v** representa a solução. Por outro lado se a matriz **M**´ ficar vazia depois de eliminar uma coluna ou se **M**´ contiver uma linha sem valores 1 e 0, então é impossível encontrar a solução neste ramo da árvore de pesquisa. Portanto é necessário retroceder até ao último ponto de ramificação no qual o exame da variável de decisão não foi concluido, inverter o valor desta variável e continuar a percorrer a árvore de pesquisa. Caso ao percorrer a árvore completa não seja possível encontrar o vector **v** ortogonal a cada linha da matriz **M** então a função booleana correspondente é insatisfazível.

Consideremos um exemplo. Vamos verificar se a função booleana seguinte é satisfazível:

$$f(x_1,\ldots,x_8) = (x_1 \vee \overline{x}_8) \wedge (x_1 \vee x_3) \wedge \wedge (x_1 \vee \overline{x}_2 \vee x_7 \vee x_8) \wedge (x_1 \vee \overline{x}_7 \vee x_8) \wedge \wedge (x_1 \vee x_2 \vee \overline{x}_3 \vee x_8) \wedge (\overline{x}_1 \vee x_2 \vee x_8) \wedge \wedge (\overline{x}_1 \vee \overline{x}_7 \vee x_8) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_7) \quad (2)$$

A função pode ser transformada na seguinte matriz **M**:

$$\mathbf{M} = \begin{array}{c} \begin{array}{cccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{array} \\ \left[\begin{array}{cccccccc|c} 1 & - & - & - & - & - & - & 0 & 1 \\ 1 & - & 1 & - & - & - & - & - & 2 \\ 1 & 0 & - & - & - & - & 1 & 1 & 3 \\ 1 & - & - & - & - & - & 0 & 1 & 4 \\ 1 & 1 & 0 & - & - & - & - & 1 & 5 \\ 0 & 1 & - & - & - & - & - & 1 & 6 \\ 0 & - & - & - & - & - & 0 & 1 & 7 \\ 0 & 0 & - & - & - & - & - & - & 8 \end{array}\right] \end{array}$$

A aplicação do algoritmo considerado à matriz **M** pode ser representada com a árvore de pesquisa seguinte (fig. 2).

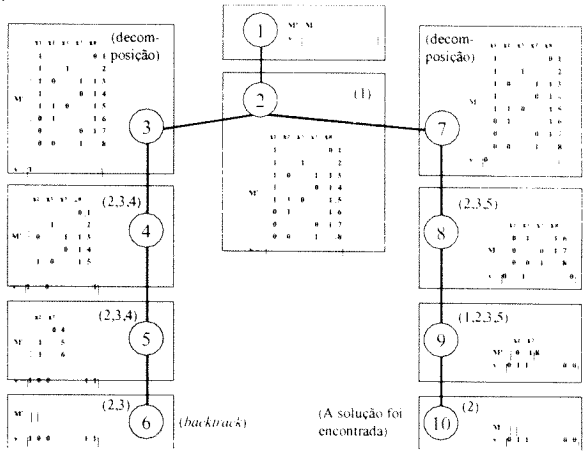


Fig. 2 – Árvore de pesquisa para encontar o vectro **v** ortogonal a cada linha da matriz **M**

Esta árvore é bastante simples e contem um só ponto de ramificação. A numeração de vértices reflecte a ordem por que são examinadas as situações intermédias. A fig. 2 mostra também a evolução do vector **v** e das matrizes **M**´ no processo de pesquisa. Entre parenteses estão indicados os métodos aplicados em cada passo do algoritmo. É fácil verificar que o vector **v** encontrado é ortogonal a cada linha da matriz **M**. Sendo assim a função (2) é satisfazível com a seguinte atribuição de valores às variáveis: x_1=1, x_2=0, x_3=0, x_7=1 e x_8=1.

III. ARQUITECTURA

Como foi mencionado na introdução, o tempo de compilação do circuito de hardware limita significativamente a gama de problemas para os quais a utilização da FPGA pode ser considerada razoável em comparação com a solução em software. Tendo isto em atenção tentámos criar um circuito universal que possua uma estrutura predefinida que pode ser reutilizada de problema a problema. Como resultado foi proposta a arquitectura representada na fig. 3.

A arquitectura contem uma unidade de controlo central que executa o algoritmo da fig. 1. A matriz é realizada com base em RAM. Cada elemento da matriz m_{ij} é codificado de maneira seguinte:

- 00 – significa zero lógico;
- 01 – significa um lógico;
- 10 – denota *don't care*;
- 11 – indica que o elemento respectivo não é utilizado, por exemplo este foi eliminado da matriz.

Note-se que esta realização permite recolher uma linha da matriz em um só ciclo de relógio. Por outro lado para ler uma coluna precisamos de aceder a cada linha, extrair dela uns determinados bits e finalmente compor destes bits a coluna.

A ALU é utilizada para calcular o número de zeros, uns e *don't cares* num vector ternário (i.e. em várias linhas e colunas da matriz – para os métodos 1, 4 e 5 na fig. 1) e verificar se dois vectores ternários são ortogonais (i.e. se alguma linha da matriz é ortogonal ao vector **v** – para o método 2 na fig. 1).

O bloco *Registos* guarda a informação seguinte:

- *cur_row* – a linha activa da matriz;
- *cur_col* – a coluna activa da matriz;
- *cur_v* – o valor actual do vector **v**;
- *cur_d* – o valor actual da variável de decisão corrente;
- *del_rows* – vector que indica as linhas apagadas da matriz;
- *del_cols* - vector que indica as colunas apagadas da matriz.

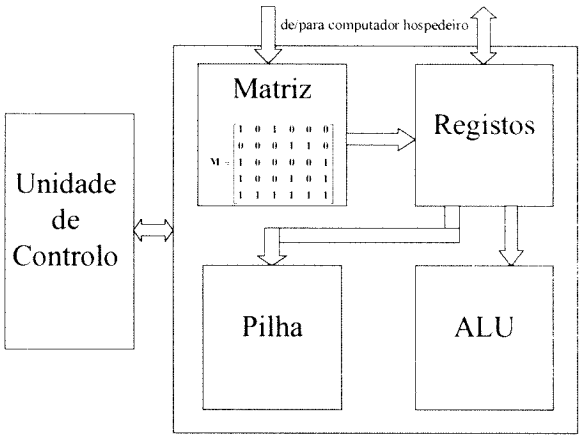


Fig. 3 – Arquitectura do circuito

A pilha é utilizada para suportar o processo de retrocesso na árvore de pesquisa (*backtrack*). Quando decompomos alguma situação, os valores de todos os registos são escritos na pilha e quando retrocedemos na árvore de pesquisa estes são recuperados da pilha para os registos.

Durante a execução do algoritmo a matriz não é modificada. Todas as alterações possíveis (eliminação de linhas e colunas) são armazenadas em registos correspondentes. Sendo assim, não é necessário guardar na pilha as matrizes intermédias.

As dimensões máximas da matriz são fixas: $m_{max}*n_{max}$. Para o caso de necessidade do processamento de uma matriz com dimensões menores existem dois registos especiais que guardam as dimensões actuais da matriz $m_{act}*n_{act}$. De acordo com estes valores a unidade de controlo só vai forçar o processamento da área da matriz pretendida.

IV. INTERACÇÃO DE SOFTWARE E HARDWARE RECONFIGURÁVEL

A arquitectura descrita na secção III satisfaz a restrição imposta: esta não é alterada de instância a instância e só precisa de receber vários dados para a matriz **M**. Um outro problema importante é que é impossível guardar e processar em FPGA uma matriz de dimensões arbitrárias. Portanto propomos utilizar a estratégia seguinte. Implementamos o algoritmo considerado numa aplicação de software. No processo de construção da árvore de pesquisa são empregues vários métodos de decomposição e de redução. Como resultado, ao se mover de cima para baixo ao longo de um dos ramos da árvore, as dimensões iniciais da matriz vão diminuindo. Logo que as dimensões novas satisfaçam as restrições do circuito (tais como o número máximo de linhas e colunas da matrix $m_{max}*n_{max}$) a matriz poderá ser transferida para a FPGA para o processamento posterior.

Esta estratégia está representada na fig. 4. Inicialmente, a aplicação de software deve configurar a FPGA com o circuito respectivo. A seguir, se as dimensões da matriz original satisfazem as restrições predefinidas, então esta poderá ser transferida para a FPGA e o problema inteiro será resolvido em hardware. Em caso oposto a aplicação de software vai resolvendo o problema até que as dimensões da matriz intermédia (que deve ser construída no passo corrente do algoritmo) satisfaçam as restrições. A partir daí a FPGA ficará responsável por todos os passos seguintes. Se o hardware reconfigurável encontrar a solução o problema está resolvido e o resultado será transferido para o computador hospedeiro. Por outro lado, se o ramo da árvore de pesquisa considerado não permitir encontrar a solução, o controlo regressará à aplicação de software que vai continuar a percorrer a árvore de pesquisa até atingir um outro ponto em que a matriz intermédia satisfaz as restrições. Os dados da matriz serão então transferidos para a FPGA e esta vai

tentar resolver o sub-problema. Os passos considerados são repetidos até encontrar a solução ou concluir que não existe solução nenhuma.

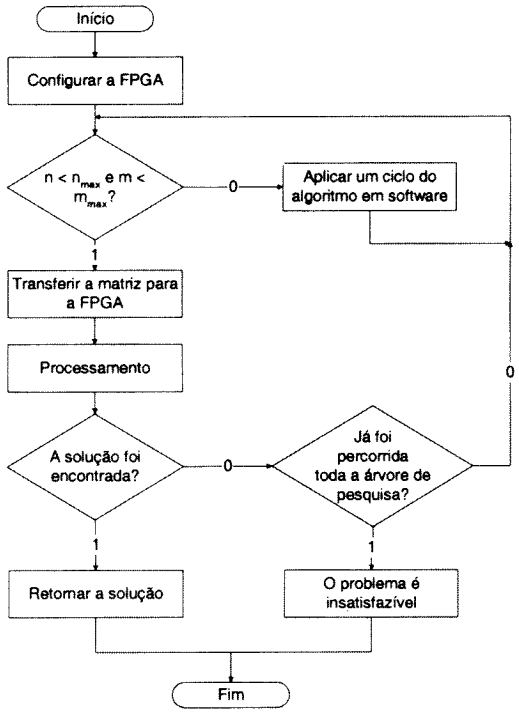


Fig. 4 – Colaboração de software e hardware reconfigurável

V. EXPERIÊNCIAS

Para as experiências foi utilizada a placa com interface PCI ADM-XRC da AlphaData [8] que contém uma FPGA XCV812E [9]. Esta FPGA é composta por um *array* de 56*84 CLBs e incorpora uma memória adicional organizada em blocos distribuidos por todo o dispositivo. Portanto, esta FPGA serve muito bem para a aplicação considerada porque podemos aproveitar a grande quantidade da memória disponível para guardar a matriz e organizá-la de maneira a assegurar a largura de banda necessária no acesso às linhas da matriz. A interacção com a FPGA é feita com a ajuda da biblioteca de interface com a placa ADM-XRC que suporta a inicialização e configuração da FPGA, transferência de dados, processamento de interrupções e erros, gestão de relógio, etc.

Foram implementados 3 circuitos com as seguintes dimensões máximas da matriz em FPGA: 64*64, 128*128 e 256*256 (vamos referenciar estes circuitos como *c64*, *c128* e *c256* respectivamente). Devido ao facto de cada elemento da matriz ser codificado com dois bits, podemos processar matrizes ternárias com as dimensões ≤64*32, ≤128*64 e ≤256*128. Na tabela 1 está representada a informação sobre a área ocupada e a frequência de relógio de cada um dos circuitos implementados. A área está expressada em número de *slices*, sendo cada CLB composto por dois *slices*.

Circuito	Área ocupada (*slices*)	% dos recursos da XCV812E	Frequência do relógio (MHz)
c64	1694	18	40.0
c128	3213	34	40.6
c256	6376	67	33.7

Tabela 1 – Parâmetros de circuitos implementados

Para estimar a eficiência da abordagem proposta realizamos uma série de experiências. Para tal escolhemos o problema *pigeon hole* do DIMACS [10]. Este problema consiste em determinar se é possível pôr $n+1$ bolas em n buracos sem ter duas bolas no mesmo buraco. Obviamente, isto é impossível, portanto, todas as instâncias são insatisfazíveis. Os parâmetros principais das cinco instâncias disponíveis no conjunto de *benchmarks* do DIMACS estão representados na tabela 2.

Instância	Nº de variáveis	Nº de cláusulas	t_{GRASP} (s)
Hole6	42	133	0.14
Hole7	56	204	4.31
Hole8	72	297	51.574
Hole9	90	415	531.961
Hole10	110	561	5685.6

Tabela 2 – Parâmetros de várias instâncias do problema *pigeon hole*

Este problema requer consumos significativos de tempo quando resolvido em software. As tabelas 3, 4 e 5 contêm os resultados da resolução do problema com a ajuda de uma aplicação desenvolvida em C++ que colabora de acordo com a estratégia descrita na secção IV com os circuitos *c64*, *c128* e *c256* respectivamente implementados em FPGA. Podemos ver que as dimensões iniciais da matriz para cada instância excedem as capacidades dos circuitos *c64* e *c128*. Portanto, nestes casos cada problema é primeiro processado em aplicação de software. Logo que uma matriz intermédia satisfaça as restrições predefinidas do circuito respectivo, esta será transferida para a FPGA e processada lá. As colunas direitas das tabelas 3, 4, e 5 indicam quantas vezes é necessário utilizar a FPGA para cada instância e circuito. Como podemos ver, à medida que a área reservada para a matriz em hardware cresce, aumenta a parte da árvore de pesquisa que é processada em hardware (ver fig. 5) e o número de transferências da matriz para a FPGA diminui.

No nosso caso o tempo de resolução de um problema é:

$$t_{total} = t_{config} + t_{sw} + t_{hw}$$

t_{config} é o tempo de configuração da FPGA com o circuito pretendido. Este varia de 0.31 s para circuito *c64* a 0.35 s para circuito *c256*, e começando com a instância *hole7* torna-se insignificante comparando-o com o tempo de execução em software "puro".

Instância	Dimensões iniciais da matriz	t_{congif} (s)	t_{sw} (s)	t_{hw} (s)	t_{total} (s)	Aceleração	FPGA
Hole6	133 * 42	0.31	0.0985	0.2205	0.629	0.223	60
Hole7	204 * 56		0.79	1.554	2.654	1.624	420
Hole8	297 * 72		7.54	12.419	20.269	2.544	3360
Hole9	415 * 90		73.912	111.442	185.664	2.865	30240
Hole10	561 * 110		879.056	1035.468	1914.834	2.969	302400

Tabela 3 – Resultados das experiências com a arquitectura *c64*

Instância	Dimensões iniciais da matriz	t_{congif} (s)	t_{sw} (s)	t_{hw} (s)	t_{total} (s)	Aceleração	FPGA
Hole6	133 * 42	0.31	0.011	0.103	0.424	0.330	4
Hole7	204 * 56		0.125	0.728	1.163	3.706	28
Hole8	297 * 72		1.195	5.838	7.343	7.024	224
Hole9	415 * 90		11.678	52.819	64.807	8.208	2016
Hole10	561 * 110		146.125	501.988	648.423	8.768	20160

Tabela 4 – Resultados das experiências com a arquitectura *c128*

Instância	Dimensões iniciais da matriz	t_{congif} (s)	t_{sw} (s)	t_{hw} (s)	t_{total} (s)	Aceleração	FPGA
Hole6	133 * 42	0.35	0.00204	0.09276	0.4448	0.315	1
Hole7	204 * 56		0.00278	0.16302	0.5158	8.356	1
Hole8	297 * 72		0.14842	2.41928	2.9177	17.676	14
Hole9	415 * 90		1.62572	21.79388	23.7696	22.379	126
Hole10	561 * 110		17.36361	218.48619	236.1998	24.071	1260

Tabela 5 – Resultados das experiências com a arquitectura *c256*

t_{sw} inclui o tempo de resolução da parte do problema em software (a parte superior da árvore de pesquisa na fig. 5), o tempo necessário para transferência da matriz para a FPGA e para ler o resultado da FPGA. t_{hw} é o tempo de resolução da parte do problema em hardware (a parte inferior da árvore de pesquisa na fig. 5). A aplicação de software foi executada num AMD Athlon/1GHz/256MB com o sistema operativo Windows2000.

Efectuamos uma comparação dos nossos resultados com os conseguidos em GRASP [5] que é um dos mais eficientes algoritmos de resolução de SAT. O GRASP foi também executado num AMD Athlon/1GHz/256MB com o sistema operativo Windows2000 e o tempo necessário para resolver cada instância do *pigeon hole* está indicado na tabela 2. A aceleração conseguida é dada por expressão seguinte: t_{GRASP}/t_{total}. Devido ao facto do circuito em FPGA ser especializado para as dimensões dos dados utilizados pelo algoritmo, as operações são executadas muito mais rápida e eficientemente do que em software. Como resultado, com o crescimento das dimensões máximas da matriz em FPGA, observa-se o aumento da aceleração da nossa implementação em comparação com o GRASP (fig. 6).

Na fig. 7 está demonstrado como dependem t_{sw}, t_{hw} e t_{total} das dimensões da matriz em FPGA. O gráfico foi preparado para a instância *hole9* mas a mesma tendência é válida para todos os outros exemplos (tabelas 3, 4 e 5).

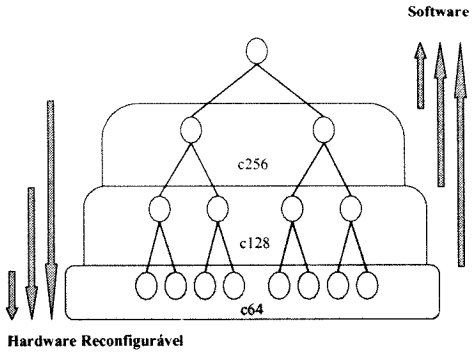


Fig. 5 – Processamento da árvore de pesquisa em software e em hardware

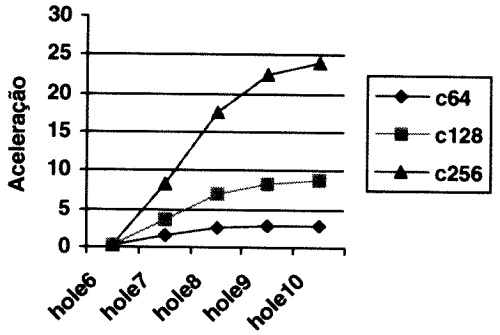


Fig. 6 – Aceleração conseguida com as três arquitecturas implementadas em comparação com o GRASP

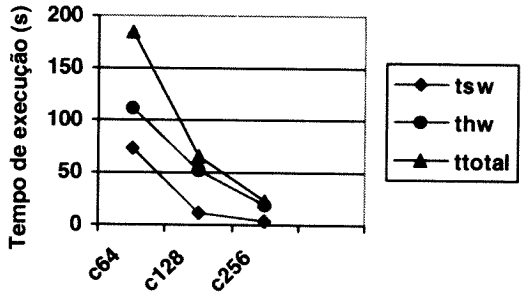


Fig. 7 – Dependência do tempo de resolução da instância *hole9* do tamanho da matriz em FPGA

Na fig. 8 está apresentada a comparação da nossa arquitectura *c256* com a arquitectura descrita em [2, 11] que usa a abordagem optimizada para cada instância do problema.

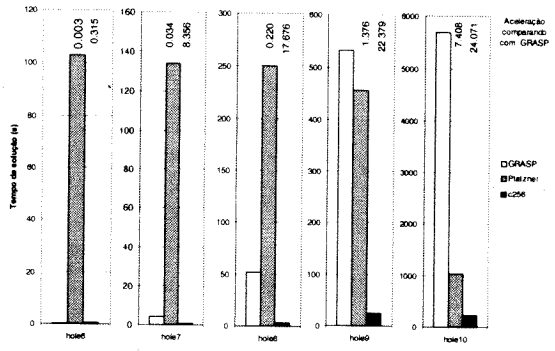


Fig. 8 – Comparação com a arquitectura proposta em [2, 11]

A frequência do relógio do circuito em [2, 11] é de 20 MHz. O tempo de resolução de cada tarefa é igual a:

$$t = t_{hc} + t_{he}$$

onde t_{hc} é o tempo de compilação do circuito de hardware (este domina em todas as instâncias consideradas) e t_{he} é o tempo de execução em FPGA. O tempo t e a aceleração em comparação com o GRASP foram reproduzidos de [11]. Contudo, em [11] o GRASP foi executado num P-II/300MHz/128MB com o sistema operativo Linux. Parece-nos no entanto não ser possível efectuar uma comparação mais exacta porque ao mudar de plataforma de software, o tempo de compilação do hardware também será modificado.

VI. CONCLUSÕES

Neste artigo apresentamos uma arquitectura para resolução de problemas de SAT baseada na colaboração de software e hardware reconfigurável. A técnica proposta permite resolver problemas com dimensões maiores do que a capacidade disponível na plataforma de hardware. Como resultado conseguimos obter uma aceleração até 24x em comparação com o GRASP. Isto é explicado pelas razões seguintes:

- o algoritmo considerado requer a execução de operações simples sobre dados regulares, permitindo a construção de uma ALU optimizada para a aplicação.
- a organização da memória onde é guardada a matriz é personalizada para os tamanhos de dados do problema.

Note-se que uma desvantagem da arquitectura proposta é que para ler uma coluna da matriz são necessários muitos acessos à memória, como é explicado na secção III. Portanto, o trabalho futuro será concentrado em resolver este problema e, em consequência, melhorar significativamente o desempenho dos circuitos.

REFERÊNCIAS

[1] D.Abramson et al., "FPGA Based Custom Computing Machines for Irregular Problems", Proc. of the HPCA98, Fevereiro de 1998, Las Vegas, Nevada.

[2] M.Platzner, "Reconfigurable Accelerators for Combinatorial Problems", IEEE Computer, pp. 58-60, April de 2000.

[3] P.Zhong, et al, "Solving Boolean Satisfiability with Dynamic Hardware Configurations", Proc. of the FPL'1998.

[4] M.Abramovici, J.T.de Sousa, "A SAT Solver Using Reconfigurable Hardware and Virtual Logic", Journal of Automated Reasoning, Fevereiro de 2000.

[5] J.M.Silva, K.A.Sakallah, "GRASP – A New Search Algorithm for Satisfiability", Proc. of the Int. Conference on CAD, pp. 220-227, Novembro de 1996.

[6] A.D.Zakrevski, "Logical Synthesis of Cascade Networks", Moscow: Nauka, 1981 (em russo).

[7] I.Skliarova, A.B.Ferrari, "Modelos matemáticos e problemas de optimização combinatória", Electrónica e Telecomunicações, vol.3, Nº3, Janeiro de 2001, pp. 202-208.

[8] http://www.alphadata.co.uk

[9] Xilinx, "The programmable Logic Data Book", San Jose, 2000.

[10] http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html

[11] O.Mencer, M.Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment", Proc. of HICSS-32, 1999.

EaSys – Uma Linguagem Orientada por Objectos para Descrição de Sistemas Digitais*

Arnaldo Oliveira, Valery Sklyarov, António Ferrari

Resumo – **Este artigo discute a utilização de linguagens de programação orientadas por objectos no projecto de sistemas digitais. Os conceitos aqui abordados são particularmente úteis para desenvolver sistemas complexos compostos por componentes de hardware e de software. A linguagem EaSys descrita neste artigo foi concebida para permitir o uso de apenas uma linguagem ao longo de todo o fluxo de projecto. Esta linguagem é uma extensão ao C++, implementada através de uma biblioteca de classes, que adiciona à linguagem base um conjunto de abstracções e mecanismos úteis para a modelação de hardware. Para escrever, compilar e depurar um modelo de um sistema escrito em EaSys, são necessárias apenas ferramentas standard de desenvolvimento em C++.**

Abstract – **This paper discusses the use of object-oriented programming languages in the design of digital systems. The ideas presented here are particularly useful to develop complex systems composed of hardware and software components. The EaSys language described in this paper was developed to allow the use of a single language in the entire design flow. This language is an extension to the C++, implemented through a class library, that adds to the base language a set of useful abstractions and mechanisms for hardware modelling. To write, compile and debug a system model written in EaSys, only standard C++ development tools are needed.**

I. INTRODUÇÃO

A evolução contínua dos processos de fabricação de circuitos integrados VLSI (*Very Large Scale Integration*) tem possibilitado a construção de dispositivos cada vez mais complexos, de menor dimensão e com melhor desempenho. A tecnologia actual permite já a implementação de Sistemas Integrados (*Systems-on-Chip – SoC*), os quais incluem numa única pastilha de silício um elevado número de componentes, tais como microprocessadores de uso geral, processadores digitais de sinal, memórias e circuitos específicos da aplicação. Assim, o processo de projecto de um SoC não consiste apenas no desenvolvimento do hardware, mas também do software que irá executar nos seus elementos de processamento programáveis. A execução de projectos desta complexidade com uma quantidade aceitável de recursos humanos e materiais requer uma produtividade elevada para que possa ser concluída com sucesso e em tempo útil. É precisamente este problema de produtividade que está na base das recentes mudanças ocorridas ao nível dos paradigmas usados no projecto de sistemas complexos e que se caracterizam essencialmente por:

- Utilização de componentes pré-projectados e completamente testados. A sua complexidade pode variar desde simples portas lógicas ou circuitos de aritmética a módulos complexos de processamento, como núcleos de processadores. Este procedimento é também vulgarmente designado por reutilização da propriedade intelectual (*Intellectual Property – IP*);
- Especialização de arquitecturas e plataformas genéricas predefinidas, para as quais existe um conjunto completo de ferramentas de desenvolvimento. Exemplos desta abordagem são a implementação de sistemas digitais em dispositivos lógicos programáveis de elevada capacidade como as FPGAs (*Field Programmable Gate Arrays*) e a optimização da implementação de um microcontrolador cuja arquitectura base é fixa mas em que os dispositivos de interface podem ser específicos da aplicação alvo;
- Adopção de métodos formais e uso de linguagens de especificação para construir na fase inicial do projecto um modelo abstracto e executável do sistema. Este modelo é iterativamente validado e refinado até à implementação. A linguagem usada deve também suportar eficientemente a descrição quer do hardware quer do software do sistema simplificando o projecto concorrente de todos os seus componentes.

Este artigo é dedicado à discussão de problemas e apresentação de soluções relacionadas com o último ponto. Além desta introdução, este artigo possui mais sete secções. Na secção II são discutidos alguns dos tópicos mais importantes do projecto de sistemas digitais complexos, tais como a utilização de especificações executáveis, as metodologias normalmente empregues e o uso de linguagens de programação no seu desenvolvimento. A descrição da linguagem EaSys e do respectivo núcleo de simulação é feita na secção III. Na secção IV são mostrados alguns exemplos de

* Este trabalho foi financiado pela bolsa de doutoramento SFRH/BD/3184/00 da Fundação para a Ciência Tecnologia.

componentes de hardware modelados com esta linguagem e na secção V descrito o seu processo de compilação e simulação. Na secção VI é feita uma comparação entre a linguagem EaSys e o SystemC. A extensibilidade da linguagem e a enumeração de alguns tópicos de trabalho futuro são tratadas na secção VII. Finalmente, as conclusões encontram-se na secção VIII.

II. PROJECTO DE SISTEMAS DIGITAIS COMPLEXOS

Nesta secção vão ser discutidas algumas das questões e problemas relacionados com o projecto de sistemas digitais complexos, nomeadamente, as vantagens da elaboração de especificações executáveis, as metodologias usadas no desenvolvimento de sistemas compostos por componentes de hardware e de software e a utilização de linguagens de programação orientadas por objectos ao longo de todo o fluxo de projecto deste tipo de sistemas.

A. Especificações Executáveis

A especificação inicial de um sistema, isto é, a descrição das suas características, comportamento e restrições é feita na maior parte dos casos em linguagem natural. Este método pode ser utilizado para descrever informalmente um sistema a projectar. Em sistemas simples pode também servir de base ao seu desenvolvimento. No entanto, no caso de sistemas complexos, o seu uso para efeitos de desenvolvimento levanta alguns problemas. Por não ser formal, uma descrição de um sistema em linguagem natural pode ser ambígua, incompleta, imprecisa e inconsistente, além de não poder ser verificada de uma forma sistemática. Estes problemas nem sempre são facilmente detectáveis numa fase inicial do projecto, que é quando a sua correcção é mais fácil e económica. Para os evitar, no início do fluxo de projecto, pode ser elaborada a partir da descrição inicial uma especificação executável, isto é, um modelo abstracto do sistema, implementado através de uma aplicação de software estruturada de acordo com um estilo bem estabelecido e cujo comportamento é o mesmo do sistema a projectar.

É importante notar que uma especificação executável não é apenas um conjunto de modelos parciais do sistema, construídos para validar determinados aspectos específicos, como por exemplo, os algoritmos utilizados. Uma especificação executável deve integrar num único modelo de alto-nível, toda a funcionalidade do sistema e as suas características mais importantes, tais como certas restrições temporais. A sua elaboração de acordo com um estilo bem estabelecido visa facilitar o seu desenvolvimento e utilização por diferentes pessoas.

As linguagens tradicionalmente utilizadas para escrever uma especificação executável são o C, o C++ [1] e o JAVA [2]. Esta escolha deve-se essencialmente a duas razões:

- Uma parte significativa do sistema é implementada em software escrito numa destas linguagens, o que facilita a sua integração com a especificação;
- Estas linguagens são mais poderosas que as linguagens de descrição de hardware (*Hardware Description Languages – HDLs*) tradicionais pemitindo escrever modelos mais compactos e a um nível de abstracção mais elevado. Como veremos mais à frente, o C++ e o JAVA podem também ser extendidos de forma a incluir as capacidades de descrição de hardware disponíveis nas HDLs.

A construção de uma especificação executável possui algumas vantagens muito importantes:

- O processo de desenvolvimento do programa ajuda a descobrir inconsistências e erros da descrição inicial do sistema. O teste do programa ajuda a perceber se a especificação inicial é ou não completa;
- Garante a ausência de ambiguidades na interpretação da especificação inicial do sistema. Sempre que surgir uma dúvida durante a implementação, a especificação pode ser executada para determinar qual deve ser o comportamento do sistema;
- Ajuda a validar a funcionalidade do sistema e os algoritmos utilizados antes de se iniciar a implementação;
- Facilita a criação de modelos iniciais de desempenho e a avaliar a eficiência do sistema;
- Possibilita a reutilização do conjunto de testes usados para validar a especificação executável, os quais podem ser refinados ou usados directamente para testar a implementação, permitindo reduzir consideravelmente o tempo dispendido em tarefas de verificação e teste.

Apesar destas importantes vantagens, uma especificação executável por si só permite solucionar apenas as questões relacionadas com a descrição e verificação iniciais do projecto de um sistema complexo. Para resolver o problema de produtividade referido acima, uma especificação executável deve ser integrada numa metodologia iterativa de descrição, verificação, refinamento e síntese suportada por ferramentas de computador para automação do projecto de sistemas electrónicos (*Electronic Design Automation – EDA*). A especificação executável actua como um modelo comportamental do sistema que é progressivamente transformado e validado até à implementação. Esta abordagem é uma generalização, para níveis de abstracção mais elevados, da síntese de hardware a partir de descrições comportamentais e RTL (*Register Transfer Level*) realizada actualmente e que se tem mostrado bastante eficaz no aumento da produtividade e, na maioria dos casos, na melhoria da qualidade final do sistema projectado.

B. Metodologias

Após a elaboração da especificação executável, o desenvolvimento de um sistema constituído por uma mistura de componentes de hardware e de software pode

ser em geral realizado segundo uma de duas metodologias distintas:

- Metodologia Multi-Linguagem (*Multi-Language Methodology – MLM*) – utiliza em cada etapa ou conjunto de etapas do fluxo de projecto a linguagem mais adequada, pelo que sempre que houver uma mudança de linguagem é necessário converter total ou parcialmente o modelo do sistema;
- Metodologia de Linguagem Única (*Single Language Methodology – SLM*) – utiliza a mesma linguagem em todas as etapas do projecto, dispensando qualquer conversão do modelo do sistema, mas à custa de uma maior versatilidade e sobrecarga da linguagem utilizada.

As metodologias MLM e SLM são descritas nas seguintes subsecções.

B.1. Metodologia MLM

As metodologias MLM são as tradicionalmente usadas no projecto de sistemas complexos compostos por componentes de hardware e de software. Na Figura 1 está representado o processo de desenvolvimento baseado numa metodologia deste tipo, onde são mostradas as etapas iniciais de simulação e refinamento do modelo inicial do sistema, bem como as etapas de simulação e síntese específicas do desenvolvimento dos seus componentes de hardware.

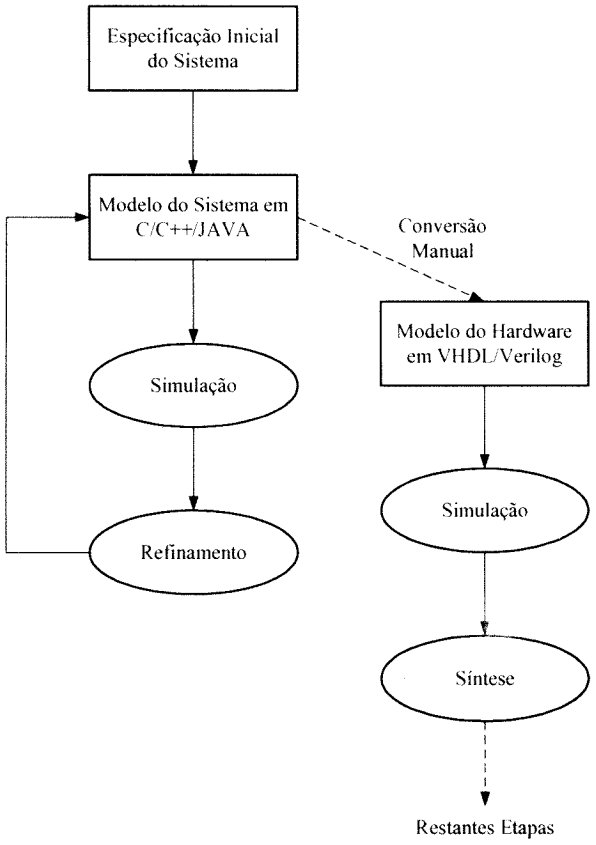


Figura 1 - Fluxo de projecto numa metodologia do tipo MLM.

Numa metodologia MLM, o ponto de partida é a especificação executável do sistema, normalmente escrita em C/C++/JAVA. Tal como foi referido na secção anterior, esta descrição actua como um modelo abstracto do sistema usado para verificar sob o ponto de vista funcional conceitos e algoritmos. Esta descrição é geralmente verificada e refinada iterativamente até se obterem os resultados esperados. Após a validação deste modelo, as partes a implementar em hardware têm que ser convertidas manualmente para um formato baseado numa HDL, normalmente VHDL [3] ou Verilog [4]. A descrição resultante é posteriormente simulada e processada pelas ferramentas de síntese e implementação específicas do hardware. Como veremos mais à frente, esta conversão é necessária porque as linguagens C/C++/JAVA não preenchem os requisitos mínimos para descrever hardware de uma forma simples, eficaz e intuitiva. As metodologias MLM, apesar de muito utilizadas, possuem alguns problemas:

- Introdução de erros durante a conversão de C/C++/JAVA para HDL – Esta conversão além de ser um processo moroso é também sujeito a erros, obrigando a que o modelo do hardware também seja validado por simulação;
- Perda de consistência entre os modelos em C/C++/JAVA e HDL – Após a conversão do modelo em C/C++/JAVA para HDL, o desenvolvimento do hardware baseia-se no refinamento e síntese do respectivo modelo. O modelo do sistema em C/C++/JAVA fica rapidamente desactualizado porque as alterações realizadas na descrição em HDL não são em geral introduzidas no modelo do sistema;
- Duplicação do esforço no desenvolvimento de testes – Os testes criados para validar o comportamento do modelo em C/C++/JAVA do sistema não podem em geral ser aplicados ao modelo HDL sem que sejam previamente convertidos. Assim, o projectista além de precisar de converter o modelo do sistema, tem também de converter o respectivo conjunto de testes.

B.2. Metodologia SLM

Para tentar solucionar os problemas das metodologias MLM têm sido propostas outras metodologias para o projecto de sistema complexos compostos por componentes de hardware e de software. Todas elas consistem na adopção de uma linguagem de programação de alto-nível com as características apropriadas para que possa ser utilizada em todo o fluxo de projecto incluindo o desenvolvimento do software e a implementação do hardware do sistema. A utilização de apenas uma linguagem desde a elaboração da especificação executável até à implementação dos componentes de hardware tem a vantagem das tarefas de simulação, refinamento e síntese serem realizadas iterativamente e sempre sobre o mesmo modelo do sistema, que é obviamente mantido consistente ao longo de todo o projecto. Assim, a conversão do

modelo em C/C++/JAVA para HDL da metodologia MLM deixa de ser necessária, a simulação é feita com um modelo escrito numa única linguagem e diferentes partes do sistema podem ser descritas a diferentes níveis de abstracção. A simulação do sistema completo é realizada num ambiente homogéneo dispensando os habituais interfaces complexos para co-simulação existentes nas ferramentas de projecto baseadas em HDLs que implementam metodologias do tipo MLM. Na Figura 2 estão representadas as etapas mais importantes de uma metodologia SLM para o caso da linguagem EaSys descrita neste artigo.

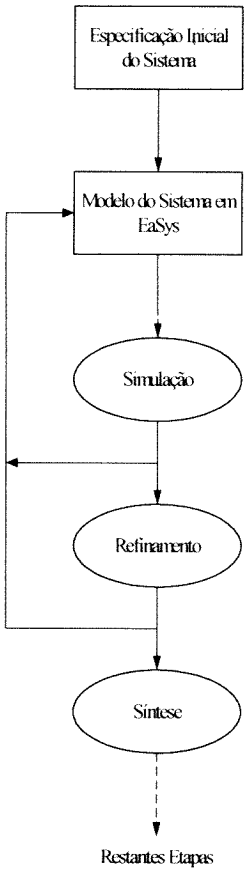


Figura 2 - Fluxo de projecto numa metodologia do tipo SLM.

Uma metodologia SLM possui relativamente a uma MLM algumas vantagens importantes:

- Refinamento progressivo – em nenhuma fase do projecto é necessário converter o modelo do sistema escrito numa linguagem para outra. O modelo é progressivamente simulado, refinado e enriquecido com detalhes desde a especificação até à implementação;
- Utilização de apenas uma linguagem – a linguagem utilizada permite não só descrever o sistema a um nível funcional abstracto mas também desenvolver o software do sistema e modelar o seu hardware ao nível comportamental, RTL ou lógico. Assim é necessário possuir apenas experiência numa única linguagem e utilizar um menor número de ferramentas de projecto;
- Reutilização dos programas de teste – os testes utilizados para validar o modelo abstracto do sistema podem ser aplicados ao modelo RTL, permitindo por um lado poupar o tempo de conversão e por outro aumentar a qualidade do processo de teste;
- Aumento da produtividade – quanto maior for o nível de abstracção de um modelo mais fácil é a sua elaboração e manutenção e a execução é mais rápida do que nos ambientes de simulação de hardware tradicionais. Além disso, tal como já foi dito, a utilização de ferramentas de síntese e implementação capazes de produzir hardware a partir de uma descrição comportamental tem-se mostrado também bastante eficaz no aumento da produtividade.

C. Linguagens

Tal como foi referido acima, a especificação executável de um sistema é normalmente feita em C/C++/JAVA porque estas linguagens além de serem as mais utilizadas no desenvolvimento de software permitem também escrever facilmente modelos a níveis de abstracção elevados. Já o hardware é tradicionalmente modelado com HDLs capazes de expressar conceitos específicos do hardware, tais como concorrência, sinais, comportamento reactivo e tipos de dados específicos deste domínio. As linguagens de programação tradicionais não suportam directamente nenhum destes conceitos, pelo que não são adequadas para descrever hardware. Por outro lado, as HDLs tradicionais foram concebidas especialmente para modelar hardware e são bastante limitativas ao nível dos paradigmas e níveis de abstracção suportados, pelo que não podem ser utilizadas no desenvolvimento de software nem na construção de modelos funcionais de alto-nível de um sistema. Por outras palavras, enquanto as linguagens de programação possuem limitações nos níveis de abstracção mais baixos normalmente usados para descrever hardware, as HDLs não são apropriadas para os níveis de abstracção mais elevados tipicamente utilizados no desenvolvimento de software. No caso de sistemas compostos tanto por componentes de hardware como de software, estas limitações levam à utilização de várias linguagens ao longo do fluxo de projecto e consequentemente a uma metodologia do tipo MLM com todos os problemas que daí resultam.

A definição de uma linguagem apropriada para ser usada numa metodologia do tipo SLM, isto é, que possa ser utilizada a vários níveis de abstracção, em todas as etapas de projecto de um sistema complexo (desde a construção da especificação ao desenvolvimento do software e projecto do hardware) e que suporte vários paradigmas de programação e modelos de computação, pode ser feita de três formas distintas:

- Concepção de uma nova linguagem capaz de lidar com todos os aspectos do projecto de um sistema digital com componentes de hardware e de software;
- Adição às HDLs de construções e das capacidades típicas das linguagens de programação;

- Extensão das linguagens de programação com os mecanismos necessários para descrever hardware.

Como será mostrado a seguir, devido à extensibilidade proporcionada pelo paradigma de orientação por objectos das linguagens C++ e JAVA, a última abordagem é mais conveniente porque a sua implementação é mais simples e a sua integração nos ambientes convencionais de especificação e desenvolvimento de software é mais directa. No entanto, antes de se discutir a extensão de uma linguagem de programação, é necesssário identificar quais os requisitos que esta deve preencher para suportar eficientemente a descrição do hardware.

C.1. Modelação de Hardware

As linguagens de programação C, C++ e JAVA foram concebidas para desenvolver software, não possuindo o suporte necessário para descrever hardware de uma forma simples, eficiente e intuitiva. Para modelar hardware com uma das linguagens referidas acima é necessário dotá-las das seguintes capacidades [5, 6]:

- Concorrência e paralelismo – o hardware é por inerência paralelo enquanto os programas de software são executados sequencialmente. A concorrência é suportada nas HDLs através da introdução da noção de processo ou tarefa. Cada tarefa executa, conceptualmente, em paralelo com os seus pares;
- Sinais e portos – no caso do software, os processos concorrentes comunicam entre si usando primitivas do tipo memória partilhada, semáforos, regiões críticas, etc. Estas primitivas assumem que cada processo pode aceder facilmente ao estado interno dos restantes, o que não acontece no hardware. Os sinais são objectos que actuam como memória partilhada para comunicação entre processos. Os portos estabelecem o interface externo de um bloco de hardware e definem o tipo de operações realizáveis sobre o sinal a que está associado;
- Comportamento reactivo – o hardware pode ser modelado como um sistema reactivo, isto é, um sistema em permanente interacção com o ambiente que o rodeia. O conceito de comportamento reactivo é fundamental para modelar hardware a todos os níveis de abtracção. Nas HDLs a implementação de comportamento reactivo baseia-se na noção de evento, isto é, uma mudança de valor de um sinal ou porto. Assim, a descrição do hardware pode ser feita com um conjunto de tarefas em execução permanente que reagem continuamente a eventos ocorridos no ambiente com o qual o sistema interage;
- Tipos de dados específicos do hardware – ao contrário do que se passa no software, em que os tipos de dados fundamentais possuem um tamanho fixo, para modelar hardware é útil dispôr de tipos de dados de tamanho parametrizável, tais como quantidades inteiras com e sem sinal de precisão arbitrária, quantidades decimais em vírgula fixa e vectores de bits para representação de valores binários e lógicos standard (alta-impedância, saídas activas/passivas e valores indefinidos para representação de conflitos em barramentos).

C.2. Extensibilidade e Orientação por Objectos

Por não suportarem os mecanismos enumerados acima, as linguagens de programação tradicionais (C/C++/JAVA) não podem ser utilizadas directamente numa metodologia do tipo SLM que cubra todas as etapas do fluxo de projecto de um sistema digital complexo. Tal como já foi dito acima, uma das abordagens para resolver este problema consiste na definição de uma linguagem de programação completamente nova, capaz de lidar não só com os aspectos de desenvolvimento do software mas também com o projecto de hardware. No entanto, esta abordagem é pouco atractiva porque requer tempos de desenvolvimento e evolução consideráveis para a linguagem e respectivas ferramentas de suporte. Além disso, os processos de adopção e aprendizagem de uma nova linguagem costumam ser bastante morosos. Felizmente, o suporte para descrever hardware pode também ser adicionado a uma das linguagens tradicionalmente usadas no desenvolvimento de software. Esta extensão pode ser feita de duas maneiras:

- Adicionando novas construções e funcionalidades que necessitem de estender a sintaxe da linguagem base e consequentemente requerem a construção de ferramentas de desenvolvimento específicas, tais como compiladores, simuladores, sintetizadores e depuradores capazes de manipular as novas construções da linguagem. Exemplos desta abordagem são o HandelC [7] e as extensões propostas para tornar as HDLs tradicionais orientadas por objectos;
- Tirando partido das capacidades de extensão disponíveis nas linguagens de programação orientadas por objectos como o C++ ou o JAVA, as quais suportam a definição de novos tipos de dados e respectivas operações. A extensão da linguagem é feita através de uma ou várias bibliotecas de classes que fornecem um conjunto de abstracções úteis para modelar os aspectos do projecto de hardware de um sistema, tais como tarefas, sinais, portos, eventos, tipos de dados e operações realizáveis sobre cada um destes objectos. Este método tem a vantagem de permitir reutilizar algumas das ferramentas disponíveis para a linguagem base, tais como compiladores, depuradores e ambientes integrados. As linguagens SystemC [8], OCAPI [9], Cynlib [10] são implementadas usando esta abordagem. No entanto, é importante referir que no caso de se pretender sintetizar hardware directamente da descrição em C/C++/JAVA continua a ser necessária a construção das respectivas ferramentas [6].

O grau de sofisticação das bibliotecas de classes, referidas no último ponto pode ser bastante variado. Numa abordagem mais elementar pode disponibilizar apenas um conjunto de abstracções úteis para modelar hardware digital ao nível lógico ou RTL e que implementam os mecanismos normalmente disponíveis nas HDLs. Estas abstracções em conjunto com as funcionalidades da linguagem base permitem realizar manual e iterativamente o refinamento e validação do modelo do sistema. Numa abordagem mais elaborada, as bibliotecas podem também disponibilizar mecanismos de comunicação e sincronização que facilitem o refinamento progressivo e sistemático do modelo inicial do sistema. Para facilitar este processo, as bibliotecas devem também possuir informação sobre como um mecanismo de comunicação ou sincronização abstracto deve ser mapeado, para efeitos de implementação, em primitivas de baixo-nível. As bibliotecas podem também possuir implementações de elementos computacionais complexos que facilitem a reutilização da IP. Para melhorar a qualidade final do sistema, os componentes disponibilizados devem possuir informação sobre a sua implementação numa dada tecnologia alvo. Para facilitar a síntese de hardware a partir de uma descrição em C++/JAVA pode definir-se um subconjunto sintetizável da linguagem a partir do qual a geração de hardware possa ser realizada automaticamente por ferramentas construídas para o efeito.

III. DESCRIÇÃO DA LINGUAGEM

A motivação para desenvolver a linguagem EaSys resultou da necessidade de conceber uma plataforma de investigação aberta, robusta, modular, flexível e extensível para especificar, verificar e sintetisar sistemas complexos compostos por uma combinação arbitrária de componentes de hardware e de software. Esta linguagem é uma extensão à linguagem de programação orientada por objectos C++, à qual foram adicionados, através de uma biblioteca de classes, os mecanismos enumerados na secção anterior, de forma a torná-la adequada não só para desenvolver software mas também para descrever hardware. Além das classes que representam conceitos do hardware, tais como módulos, tarefas, sinais e portos, a linguagem EaSys possui um núcleo de simulação baseado em ciclos, que deve ser associado ao modelo do sistema a projectar. Este núcleo disponibiliza um interface para controlo da execução do modelo do sistema durante a sua simulação.

Numa primeira fase, a linguagem EaSys adiciona ao C++ apenas as construções e mecanismos que implementam os subconjuntos sintetizáveis das HDLs tradicionais. Há medida que a linguagem for evoluindo, vão sendo disponibilizadas novas capacidades que possibitam a construção de modelos a níveis de abstracções mais elevados. É concensual que para lidar eficientemente com a complexidade crescente dos sistemas digitais, um projecto deve ser realizado usando uma abordagem do tipo decomposição descendente (*top-down decomposition*), em que um modelo de alto-nível é sucessivamente validado e refinado até à implementação. Por outro lado, a concepção das abstracções usadas em modelos de alto-nível deve seguir uma abordagem do tipo montagem ascendente (*bottom-up assembly*) de forma a garantir que possuem implementação física.

A especificação e simulação de um sistema descrito na linguagem EaSys são feitas apenas com ferramentas standard de desenvolvimento em C++, tais como editores de texto, compiladores, depuradores e ambientes integrados. Isto é possível porque todos os mecanismos da linguagem foram implementados através de definição de classes e sem estender a sintaxe da linguagem C++. No seu estado actual, a linguagem EaSys pode ser utilizada nas etapas iniciais de uma metodologia do tipo SLM. A síntese directa de hardware a partir de um modelo escrito em EaSys ainda não é suportada porque requer a construção de ferramentas capazes de interpretar a semântica das operações realizadas sobre os objectos da linguagem.

A plataforma de desenvolvimento da linguagem EaSys foi o Microsoft Visual Studio 6.0 sobre o sistema operativo Microsoft Windows 2000. No entanto, para facilitar a portabilidade para outras plataformas (e.g. Linux) houve o cuidado de usar sempre que possível as construções e as bibliotecas standard do C++. As poucas excepções a esta regra são perfeitamente localizadas e controladas por directivas de compilação condicional dependentes da plataforma. As relações de herança das classes que constituem o núcleo da linguagem EaSys estão representadas na Figura 3. Todas as classes estão implementadas na versão actual da linguagem, à excepção da *CEaSysThread* e *CEaSysBus*. As classes parametrizáveis são implementadas usando *templates* do C++. As classes da Figura 3 podem ser divididas nos seguintes grupos funcionais:

- Serviços base
  - *CEaSysObject*
- Eventos
  - *CEaSysEventBase*
- Módulos
  - *CEaSysModule*
- Tarefas
  - *CEaSysTask*
  - *CEaSysRoutine*
  - *CEaSysThread*
- Sinais
  - *CEaSysSignalBase*
  - *CEaSysSignal*
  - *CEaSysBus*
- Portos
  - *CEaSysPortBase*
  - *CEaSysPort*
  - *CEaSysInPort*
  - *CEaSysOutPort*
  - *CEaSysInOutPort*
- Núcleo de simulação
  - *CEaSysKernel*

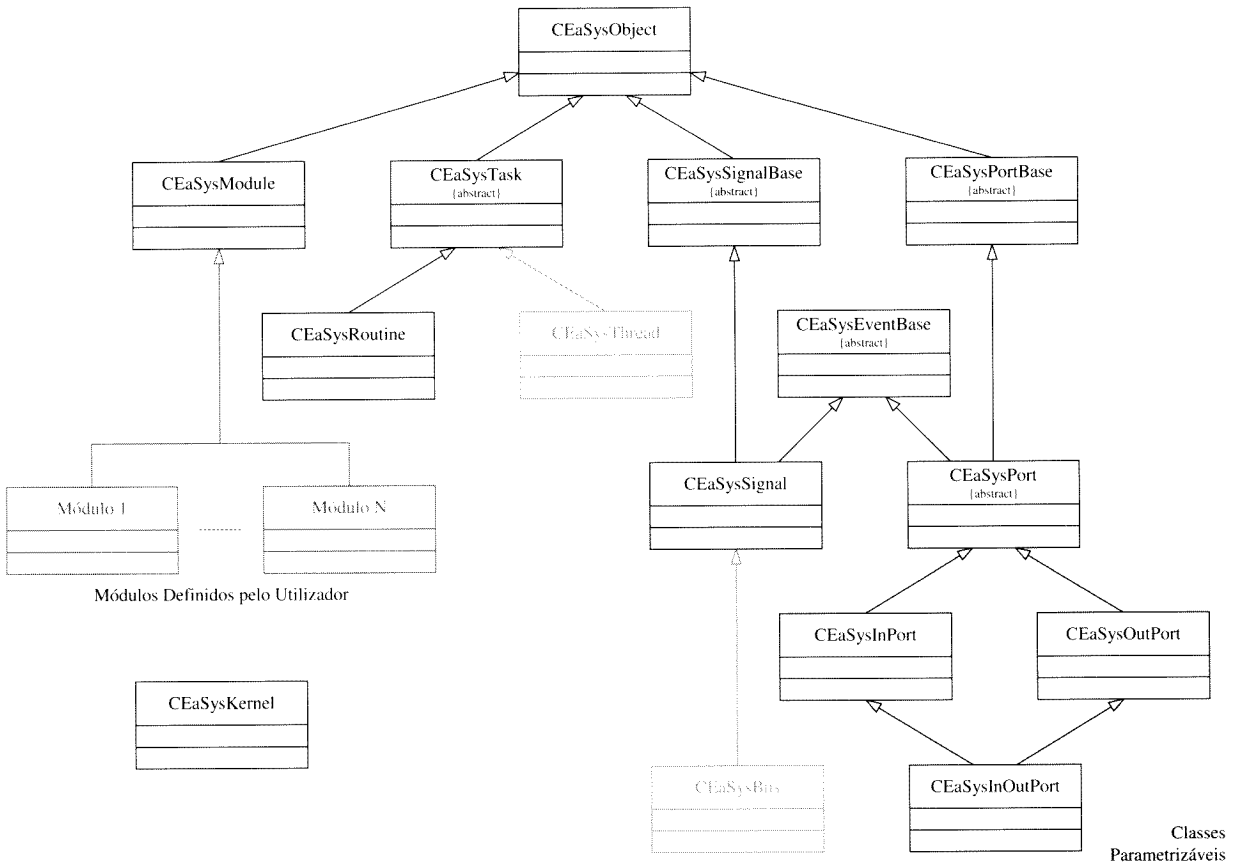


Figura 3 - Relações de herança entre as classes que implementam o núcleo da linguagem Easys.

Como veremos ao longo da descrição da linguagem, o utilizador pode usar quer os verdadeiros nomes das classes e a síntaxe normal do C++ para instanciar e manipular os respectivos objectos, quer as macros disponibilizadas para o efeito. A segunda abordagem apesar de mais restritiva é recomendável para uma utilização corrente, pois simplifica a realização das operações sobre os objectos através do encapsulamento de alguns dos detalhes associados.

As subsecções seguintes descrevem cada um dos elementos da linguagem. Esta discussão será acompanhada da apresentação progressiva de um exemplo de uma porta lógica complexa AOI representada na Figura 4. A descrição em EaSys do circuito da Figura 4 é apresentada na Figura 5. Este exemplo é interessante porque permite ilustrar as capacidades de descrição estrutural e comportamental da linguagem.

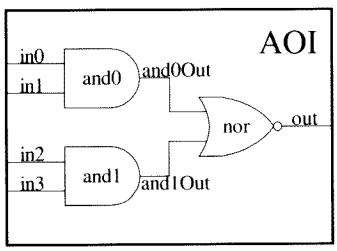


Figura 4 - Esquema de uma porta lógica complexa AOI.

```
module(AOI)
{
   iport<bool> in0;
   iport<bool> in1;
   iport<bool> in2;
   iport<bool> in3;
   oport<bool> out;

   signal<bool> and0Out, and1Out;

   method(nor)
   {
      out = !(and0Out || and1Out);
   }

   constructor(AOI)
   {
      instance(AND2, and0);
      and0->in0(in0);
      and0->in1(in1);
      and0->out(and0Out);

      instance(AND2, and1);
      and1->in0(in2);
      and1->in1(in3);
      and1->out(and1Out);

      routine(nor);
      nor->sensitive(and0Out);
      nor->sensitive(and1Out);
   }
};
```

Figura 5 - Descrição da porta lógica AOI em EaSys.

A. Serviços Base

A classe *CEaSysObject* serve de base à maioria das classes que constituem o núcleo da linguagem EaSys. Os objectos do tipo módulo, tarefa, sinal e porto devem possuir uma identificação que consiste num nome e na sua posição hierárquica dentro do sistema representada por uma referência para o objecto pai (onde é feita a instanciação). A classe *CEaSysObject* implementa os mecanismos para armazenar e manipular esta informação e disponibiliza-os às suas classes derivadas. Esta classe não pode ser instanciada directamente porque o seu construtor só está disponível por derivação.

B. Eventos

Diz-se que ocorreu um evento num objecto quando este muda de valor. Os eventos podem ocorrer em objectos do tipo sinal ou porto. Um evento pode ser positivo ou negativo dependendo do sentido da variação do valor. Os eventos são representados no núcleo da linguagem EaSys pela classe *CEaSysEventBase*. Por ser abstracta, esta classe não pode ser instanciada e define apenas um interface que os objectos, onde ocorrem eventos, devem implementar. Assim, as classes *CEaSysSignal* e *CEaSysPort* que representam, respectivamente, os sinais e os portos da linguagem devem também ser derivadas da classe *CEaSysEventBase*. Desta forma é possível realizar invocações polimórficas, isto é, independentemente se o objecto é um sinal ou um porto e do respectivo tipo de dados. Na Tabela 1 são apresentadas as construções utilizadas para testar a ocorrência de eventos em sinais e portos. A segunda e a terceira coluna da tabela ilustram, respectivamente, as construções usadas no caso do teste ser efectuado directamente sobre o objecto ou através de um ponteiro para o objecto. Estas expressões retornam verdadeiro se o respectivo evento ocorreu, ou falso em caso contrário.

Tipo do Evento	Objecto	Ponteiro
Qualquer	`obj.event`	`pObj->event`
Positivo	`obj.posevent`	`pObj->posevent`
Negativo	`obj.negevent`	`pObj->negevent`

Tabela 1 - Construções usadas para testar a ocorrência de eventos em objectos do tipo sinal ou porto.

C. Módulos

Os módulos são os elementos básicos de decomposição estrutural da linguagem EaSys. Estes permitem dividir um sistema em blocos mais simples, logo mais fáceis de modelar, desenvolver, testar e integrar. No caso de projectos complexos permitem também fazer a sua distribuição entre os vários projectistas da equipa de desenvolvimento. Um módulo permite esconder a representação de dados e os algoritmos internos dos restantes módulos, isto é, promovem a separação entre o interface e a implementação dos componentes de um sistema. Isto possibilita o estabelecimento de interfaces públicos e bem definidos, o que facilita a interacção entre os diversos módulos do sistema, bem como a manutenção e introdução de modificações. Por exemplo, um projectista pode decidir alterar completamente a implementação interna do módulo. Se o seu interface externo e o comportamento permanecerem inalterados, os utilizadores do módulo não se apercebem das alterações internas, permitindo assim a realização de optimizações locais.

Um módulo é declarado com a palavra-chave *module*, tal como é ilustrado no exemplo seguinte:

```
module(AOI)
{
   ...
};
```

O identificador após a palavra-chave "*module*" é o nome do modulo, que neste caso é "AOI". Como veremos mais à frente, um módulo é na realidade uma classe em C++ derivada da classe "*CEaSysModule*" definida no núcleo da linguagem. A palavra-chave "*module*" é uma macro cujo parâmetro é o nome da classe que representa o módulo. Assim, em vez de se utilizar a macro "*module*", um módulo pode ser declarado usando a sintaxe usual do C++ para derivação de classes, ou seja:

```
class AOI : public CEaSysModule
{
   ...
};
```

A declaração de um módulo usando a macro "*module*" tem a vantagem de ser simultaneamente mais legível e mais concisa. Um módulo pode conter outros tipos de objectos, tais como portos, sinais internos, variáveis locais, tarefas e outros módulos. Além disso, todos os módulos devem possuir um construtor onde são declaradas as tarefas e/ou definida a sua estrutura interna. Em conjunto, estes objectos implementam a sua funcionalidade. O construtor de um módulo pode ser declarado da seguinte maneira:

```
module(AOI)
{
   ...
   constructor(AOI)
   {
      ...
   }
   ...
};
```

O identificador entre parêntesis a seguir à palavra-chave "*constructor*" é o nome da classe do módulo ao qual pertence o construtor. Este é invocado sempre que for criada uma instância desse módulo. No corpo do construtor são declaradas e/ou inicializadas as estruturas de dados (módulos, tarefas, sinais, portos e variáveis) utilizadas internamente. O número e tipo de parâmetros do construtor é arbitrário. No entanto, existe uma restrição: o construtor deve obrigatoriamente possuir dois parâmetros

para receber durante a instanciação: o nome e uma referência para o objecto onde o módulo é criado. A macro *"constructor"* existe precisamente para esconder estes detalhes, simplificando a declaração do construtor.

No corpo de um construtor podem ser instanciados outros módulos usando a palavra chave *"instance"* implementada também por uma macro, cujos parâmetros são o nome (mais precisamente o tipo) do módulo a intanciar e o nome do objecto:

```
module(AOI)
{
   ...
   constructor(AOI)
   {
      instance(AND2, and1);
      ...
   }
   ...
};
```

A macro *"constructor"*pode ser usada apenas no caso dos parâmetros do construtor serem os obrigatórios acima referidos. Sempre que se utilizar outros parâmetros é necessário escrever o protótipo completo do construtor e passar ao construtor da classe base os parâmetros obrigatórios. Neste caso, a instanciação de um módulo deve também ser feita sem recorrer à macro *"instance"*. O exemplo seguinte ilustra esta situação:

```
module(AOI)
{
   ...
   AOI(type1 param1, ..., typeM paramM,
       const string& name,
       CEaSysModule* pParent) :
      CEaSysModule(name, pParent)
   {
      AND* and1= new AND(arg1, ..., argN,
                         "and1", this);
      ...
   }
   ...
};
```

O valor dos parâmetros obrigatórios referidos acima (nome e referência para o objecto pai) apesar de não afectarem o comportamento do sistema, são importantes, pois é deles que depende a construção da hierarquia do sistema e a visualização intuitiva dos resultados da simulação.

No núcleo da linguagem, a funcionalidade básica de um módulo é implementada pela classe *"CEaSysModule"*. É importante referir que esta classe destina-se apenas a servir de base às classes definidas pelo utilizador e que representam os módulos de um dado projecto ou biblioteca. Por este motivo a classe *"CEaSysModule"* não pode ser instanciada directamente.

D. Tarefas

A execução de uma aplicação de software num processador de uso geral consiste na execução sequencial das instruções que constituem o respectivo programa. As linguagens de programação tradicionais são bastante adequadas para desenvolver este tipo de aplicações, podendo também ser utilizadas facilmente para modelar o comportamento sequencial de um sistema. No entanto, os sistemas electrónicos são intrinsecamente paralelos, isto é, são compostos por vários elementos que operam em simultâneo. A modelação destas actividades paralelas com uma linguagem sequencial pode revelar-se uma tarefa bastante complexa, pouco intuitiva e sujeita a erros.

As tarefas são o elemento básico de processamento da linguagem EaSys, onde são realizadas as operações que emulam o funcionamento em paralelo dos vários elementos do sistema modelado.

As tarefas possuem listas de sensibilidade onde são especificados os eventos responsáveis pela sua activação. Uma tarefa é activada, ou seja, a sua execução é iniciada ou retomada, sempre que o valor de um sinal ou porto especificado na sua lista de sensibilidade mudar. Por defeito, isto é, na ausência de uma lista de sensibilidade, uma tarefa é executada em todos os ciclos de simulação.

As tarefas não podem ser hierárquicas, isto é, uma tarefa não pode invocar outra directamente. No entanto, uma tarefa pode provocar a activação de outra(s) através da atribuição de um valor a um sinal pertencente à(s) respectiva(s) lista(s) de sensibilidade. Além disso, as tarefas podem invocar métodos e funções ordinárias.

Existem vários tipos de tarefas. A versão actual da linguagem apenas suporta tarefas do tipo rotina. Além destas, estão também previstas tarefas do tipo *thread* que ao contrário das primeiras possuem um contexto de execução independente.

D.1. Rotinas

Este é o único tipo de tarefa suportado actualmente e que corresponde em termos funcionais aos processos da linguagem VHDL. Uma rotina é implementada por um método cuja execução é iniciada sempre que ocorrer um evento ao qual a tarefa seja sensível. Como uma rotina corre no mesmo contexto de execução da entidade invocadora (o núcleo de simulação da linguagem) é importante que, sempre que for activada, execute completamente e retorne o controlo ao núcleo de simulação. Por outras palavras, uma rotina não pode suspender a sua execução nem possuir ciclos infinitos e executa conceptualmente em tempo zero.

D.2. Contextos de execução independentes – *Threads*

Este tipo de tarefa ainda não é suportado na versão actual da linguagem. Ao contrário de uma tarefa do tipo rotina, uma do tipo *thread* possui o seu próprio contexto de execução. Isto permite que uma tarefa deste tipo possa suspender a sua execução num dado ponto. Mais tarde, quando for novamente activada, a execução é retomada a partir do ponto onde foi suspensa. Uma das aplicações das tarefas do tipo *thread* é a descrição de sistemas sequenciais sem definir explicitamente os estados e

portanto a um nível de abstracção mais elevado do que o proporcionado pelo modelo de máquina de estados finitos.

Em qualquer dos tipos de tarefa (rotina ou *thread*), o método que a implementa pertence ao módulo onde esta foi declarada. Quando é instanciado um objecto do tipo tarefa é feito o seu registo no núcleo de simulação para que a sua activação seja feita sempre que ocorrer um evento ao qual esta seja sensível. Por esta razão uma tarefa não é apenas uma rotina mas um objecto registado no núcleo de simulação e que internamente possui uma lista de sensibilidade e uma referência para o método que implementa as suas operações.

O exemplo seguinte ilustra a declaração e a definição de uma tarefa do tipo rotina chamada "nor".

```
module(AOI)
{
   ...
   method(nor)
   {
      out = !(and0Out || and1Out);
   }
   ...
   constructor(AOI)
   {
      ...
      routine(nor);
      nor->sensitive(and0Out, any);
      nor->sensitive(and1Out, any);
      ...
   }
   ...
};
```

A declaração de uma tarefa do tipo rotina é feita através da palavra-chave "*routine*" seguida do respectivo nome entre parêntesis. Esta palavra-chave é na realidade uma macro cujo parâmetro é o nome da tarefa. A sua implementação é feita no método com o mesmo nome e precedido pela palavra-chave "*method*".

A especificação da lista de sensibilidade da tarefa é também ilustrada no exemplo anterior. Neste caso, a tarefa é sensível a qualquer evento ocorrido nos sinais *"and0Out"* e *"and1Out"*. Se em vez disso se pretender que a tarefa seja sensível apenas a eventos positivos ocorridos em *"and1Out"* e a eventos negativos de *"and2Out"*, a lista de sensibilidade deveria ser definida da seguinte maneira:

```
nor->sensitive(and0Out, pos);
nor->sensitive(and1Out, neg);
```

No caso de uma tarefa ser sensível a qualquer evento de um sinal, como acontece no exemplo apresentado, o especificador "any" pode ser omitido:

```
nor->sensitive(and0Out);
nor->sensitive(and1Out);
```

A declaração de uma tarefa e a especificação da sua lista de sensibilidade devem ser feitas no corpo do construtor do respectivo módulo.

A implementação das tarefas no núcleo da linguagem EaSys consiste nas seguintes três classes:

- *CEaSysTask* – implementa a funcionalidade base de uma tarefa. Independentemente do tipo, uma tarefa deve possuir uma lista de sensibilidade onde são especificados os eventos responsáveis pela sua activação. É através desta classe que o núcleo da linguagem verifica se uma tarefa deve ou não ser activada num dado ciclo de simulação. A forma como é feita a activação é considerada específica de cada tipo de tarefa, pelo que não está definida nesta classe, devendo ser implementada nas classes derivadas desta;
- *CEaSysRoutine* – esta classe implementa os aspectos específicos de uma tarefa do tipo rotina. É através desta classe que o núcleo de simulação invoca o método que implementa a tarefa, ficando a aguardar até este concluir a sua execução;
- *CEaSysThread* – esta classe implementa os aspectos específicos das tarefas do tipo *thread*. Neste caso o controlo de execução da tarefa é mais complexo porque envolve a verificação da suspensão de execução da tarefa e a sua reactivação. Esta classe ainda não está implementada.

E. Sinais

Os sinais da linguagem EaSys são objectos especiais usados para armazenar informação e interligar portos. É também nos sinais que ocorrem os eventos responsáveis pela activação das tarefas.

Os sinais são, do ponto de vista da sintaxe, análogos às variáveis ordinárias do C++. Um sinal possui sempre um tipo de dados associado que deve ser estabelecido no momento da sua instanciação. Este, tanto pode ser um dos tipos de dados predefinidos da linguagem C++, como um tipo de dados abstracto definido pelo utilizador. No segundo caso, existem alguns requisitos que devem ser cumpridos. Em particular, os operadores relacionais (==, !=, <, >) devem estar definidos, pois são usados pelo núcleo da linguagem na detecção de eventos.

A principal diferença entre um objecto ou variável ordinária do C++ e um sinal reside na semântica das operações de leitura e escrita. Enquanto no primeiro caso uma operação de leitura devolve sempre o último valor atribuído, num sinal as alterações produzidas por uma operação de atribuição ou escrita só se tornam visíveis (i.e. devolvidas pelas operações de leitura) quando todas as tarefas activadas num dado ciclo de simulação tiverem executado. Isto permite simular a execução paralela das tarefas e detectar a ocorrência de eventos. Por este motivo, ao contrário de uma variável ou objecto ordinário, um sinal pode ser utilizado na lista de sensibilidade de uma tarefa.

No caso de um sinal ser utilizado para interligar os portos de diferentes módulos, a informação é transferida entre portos como se estes estivessem ligados directamente.

A instanciação de um sinal consiste em três elementos: a palavra-chave "signal", seguida do tipo de dados associado ao sinal especificado entre parêntesis <...> (i.e.

o parâmetro da classe) e, finalmente, o nome do objecto. Na seguinte declaração são instanciados dois sinais do tipo predefinido "bool":

```
module(AOI)
{
   ...
   signal<bool> and0Out, and1Out;
   ...
};
```

Um sinal do tipo abstracto "EState" seria declarado da seguinte forma:

```
signal<EState> currentState;
```

A palavra-chave "signal" é na realidade uma macro que cria um pseudónimo para a classe *CEaSysSignal* do núcleo da linguagem.

Para simplificar a realização de operações sobre sinais, foi redefinida, para os objectos deste tipo, a maioria dos operadores da linguagem C++. A listagem dos operadores redefinidos encontra-se na Tabela 2. Assim, o mesmo operador pode ser aplicado quer a objectos do tipo sinal, quer a objectos do tipo de dados associado ao sinal. Além disso, pode-se na mesma expressão misturar objectos de ambos os tipos. É importante referir que os operadores da Tabela 2 só estão efectivamente disponíveis se estiverem definidos para o tipo de dados associado ao sinal. Isto acontece com os tipos predefinidos do C++, mas não com os tipos de dados abstractos definidos pelo utilizador, onde é necessário definir os operadores que se pretende utilizar. Uma discussão sobre a implementação dos operadores para os tipos de dados definidos pelo utilizador está fora do âmbito deste artigo. Em [11] é feita uma descrição detalhada sobre este assunto.

A implementação dos objectos do tipo sinal no núcleo da linguagem EaSys está distribuída por três classes:

- *CEaSysSignalBase* – esta classe serve de base à parametrizável *CEaSysSignal* possibilitando a utilização dos mecanismos de polimorfismo disponíveis no C++. Esta classe é necessária, uma vez que não existe qualquer relação entre as diferentes especializações de uma classe parametrizável. Além de não ser instanciável directamente, a sua existência é completamente transparente ao utilizador;
- *CEaSysSignal* – esta classe é parametrizável, implementando, entre outras coisas, todos os operadores listados na Tabela 2. As operações definidas nesta classe são independentes do tipo de dados associado ao sinal que actua como parâmetro da classe;
- *CEaSysBus* – esta classe ainda não está implementada. Como o próprio nome indica, destina-se a fornecer uma abstracção para objectos do tipo barramento. Ao contrário dos sinais convencionais implementados pela classe anterior, um barramento pode ser controlado simultaneamente por várias saídas, pelo que o tipo de dados que lhe está associado deve representar os valores lógicos standard ('0', '1', 'L', 'H', 'Z', 'X') e necessita de mecanismos para resolver eventuais conflitos.

	Símbolo	Nome	Tipo
Leitura	(type)	Conversão	-
	==	Igualdade	Binário
	!=	Desigualdade	Binário
	<	Menor que	Binário
	>	Maior que	Binário
	<=	Menor ou igual que	Binário
	>=	Maior ou igual que	Binário
	!	NOT lógico	Unário
	&&	AND lógico	Binário
	||	OR lógico	Binário
	~	NOT bit a bit	Unário
	&	AND bit a bit	Binário
	|	OR bit a bit	Binário
	^	EXOR bit a bit	Binário
	<<	Deslocamento para a esquerda	Binário
	>>	Deslocamento para a direita	Binário
	+	Adição unária	Unário
	-	Subtracção unária	Unário
	+	Adição	Binário
	-	Subtracção	Binário
	*	Multiplicação	Binário
	/	Divisão	Binário
	%	Resto de divisão inteira	Binário
Escrita	++	Pré/Pós incremento	Unário
	--	Pré/Pós decremento	Unário
	=	Atribuição	Binário
	&=	AND bit a bit/Atribuição	Binário
	|=	OR bit a bit/Atribuição	Binário
	^=	EXOR bit a bit/Atribuição	Binário
	<<=	Deslocamento para a esquerda/Atribuição	Binário
	>>=	Deslocamento para a direita/Atribuição	Binário
	+=	Adição/Atribuição	Binário
	-=	Subtracção/Atribuição	Binário
	*=	Multiplicação/Atribuição	Binário
	/=	Divisão/Atribuição	Binário
	%=	Resto de divisão inteira/ Atribuição	Binário
	[]	Indíce de vector	-
	,	Vírgula	-

Tabela 2 - Listagem dos operadores definidos na classe parametrizável *CEaSysSignal* que define os objectos do tipo sinal da linguagem EaSys.

F. Portos

Os portos definem o interface de um módulo com o exterior, através do qual é transferida infomação. Um porto pode ser usado na lista de sensibilidade de uma tarefa, pelo que também pode disparar acções no interior do módulo. Um porto pode funcionar num dos seguintes três modos:

- Entrada - transfere dados do exterior para o módulo;
- Saída - transfere dados do módulo para o exterior;
- Entrada/Saída - pode transferir dados em ambos os sentidos dependendo da operação do módulo em cada instante.

Um porto pode estar ligado a outro porto ou a um sinal. Quando dois portos estão interligados através de um sinal, a informação é transferida entre os dois como se estes estivessem ligados directamente. Quando é realizada uma operação de escrita num porto o novo valor é atribuído ao sinal a que está directa ou indirectamente ligado. Numa operação de leitura é devolvido o valor do referido sinal.

Tal como acontece com os sinais, os valores atribuídos aos portos só se tornam visíveis através de uma operação de leitura após a conclusão do ciclo de simulação.

Na Tabela 3 estão indicadas as conexões permitidas entre os portos dos vários modos e sinais. A mesma informação está também representada graficamente na Figura 6. Na Figura 7 é ilustrado um exemplo complexo de uma hierarquia de módulos e interligação entre os seus sinais e portos. A verificação das ligações entre portos e sinais é feita a dois níveis: sintático e semântico. A verificação sintática é feita segundo as regras de compatibilidade apresentadas na Tabela 3 e no momento da compilação do modelo. A verificação semântica consiste na análise do número de portos ligados a um sinal ou porto e é realizada durante a inicialização do modelo e dinamicamente durante a sua execução. Todas as conexões da Figura 7 estão correctas, à excepção das marcadas com o símbolo "**×**" porque os portos de entrada e bidireccionais não podem ser deixados em aberto e os portos de saída e bidireccionais não podem ser controlados por mais do que uma saída ou sinal.

		Tipo do porto		
		I	*O*	*I/O*
Objecto do nível seguinte	*I (porto)*	✓	×	×
	O (porto)	✓	✓	×
	I/O (porto)	✓	✓	✓
	S (sinal)	✓	✓	✓

Tabela 3 - Ligações permitidas entre os portos de um módulo e os objectos (sinais ou portos) do nível hierárquico seguinte.

À semelhança dos sinais, a instanciação de um porto também consiste em três elementos: uma palavra reservada que depende do modo de funcionamento do porto, um tipo de dados e, finalmente, o nome do objecto. O exemplo seguinte ilustra a declaração dos portos para o exemplo da porta lógica complexa AOI:

```
module(AOI)
{
   iport<bool> in0;
   iport<bool> in1;
   iport<bool> in2;
   iport<bool> in3;
   oport<bool> out;
   ...
};
```

As palavras-chave usadas para instanciar objectos do tipo porto depedem do seu modo de operação e são: *iport* (porto de entrada), *oport* (porto de saída) e *ioport* (porto de entrada/saída – bidireccional). Os portos de um módulo não são declarados como parâmetros formais do seu construtor, porque isso representaria, em termos de escrita, uma sobrecarga considerável.

A ligação dos portos de um módulo deve ser feita logo após a sua instanciação e antes de iniciada a simulação. Assim, o sítio mais correcto para o fazer é o construtor do módulo, uma vez que é onde se define a sua estrutura. O

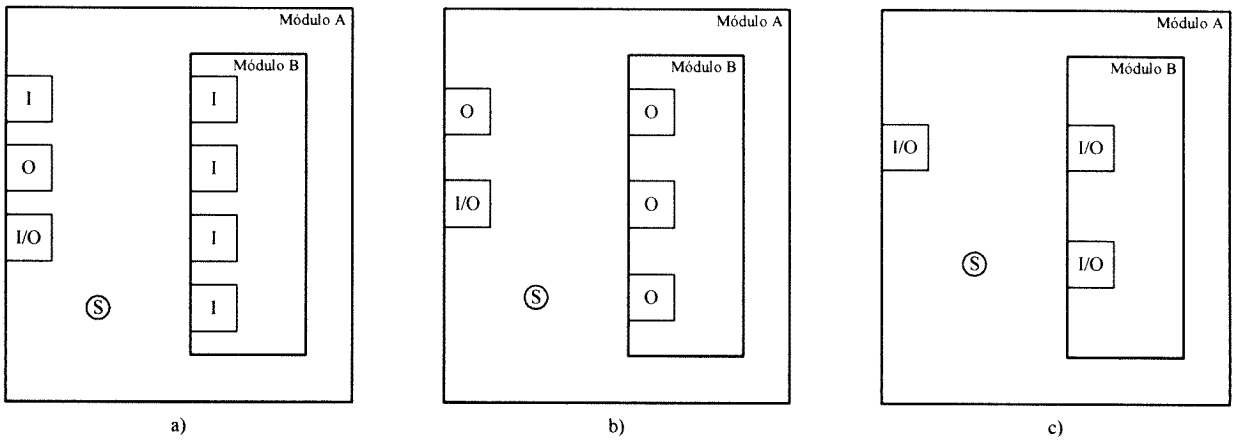


Figura 6 - Representação gráfica das ligações permitidas entre os portos de um módulo e os objectos (sinais ou portos) do nível hierárquico seguinte no caso de um: a) porto de entrada; b) porto de saída; c) porto bidireccional.

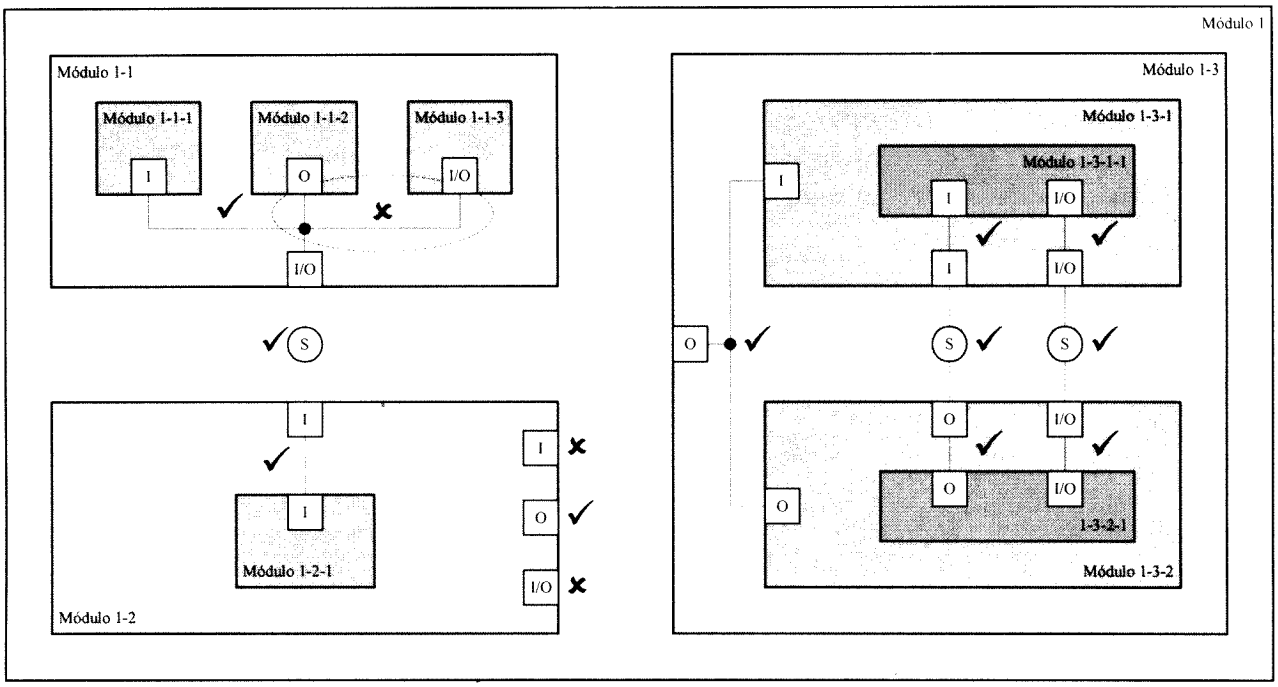


Figura 7 - Exemplo de ligações entre sinais e portos de diferentes modos.

exemplo seguinte ilustra a ligação de duas portas lógicas AND de duas entradas, quer aos portos de entrada do módulo AOI, quer a dois sinais internos.

```
module(AOI)
{
   ...
   constructor(AOI)
   {
      instance(AND2, and0);
      and0->in0(in0);
      and0->in1(in1);
      and0->out(and0Out);

      instance(AND2, and1);
      and1->in0(in2);
      and1->in1(in3);
      and1->out(and1Out);
      ...
   }
};
```

Podem também ser instanciados vectores de portos usando a sintaxe usual de declaração de vectores do C++. O exemplo seguinte ilustra esta facilidade, substituindo as quatro entradas do módulo AOI por um vector de quatro portos:

```
iport<bool> in[4];
```

Um vector de portos bidireccionais pode ser declarado da seguinte maneira:

```
ioport<bool> data[16];
```

Cada um dos portos do vector deve ser individualmente interligado e se necessário adicionado separadamente à lista de sensibilidade das tarefas que deles dependam.

A implementação dos portos na linguagem EaSys consiste nas seguintes cinco classes:

- *CEaSysPortBase* – as razões da existência desta classe são análogas às da classe *CEaSysSignalBase*. Esta classe também é transparente para o utilizador;
- *CEaSysPort* – à semelhança da classe *CEaSysSignal*, a classe *CEaSysPort* também é parametrizável, permitindo instanciar e efectuar operações sobre portos independentemente do tipo de dados subjacente. Nesta classe são implementadas as funcionalidades base de um porto que não dependam do seu modo de funcionamento (entrada, saída, entrada/saída);
- *CEaSysInPort* – esta classe é derivada da *CEaSysPort* e implementa os aspectos específicos de um porto de entrada, o qual deve obrigatoriamente estar ligado a outro porto ou sinal e sobre o qual só podem ser realizadas operações de leitura;
- *CEaSysOutPort* – esta classe também é derivada da *CEaSysPort* e implementa a funcionalidade de um porto de saída. Ao contrário do que acontece com as linguagens de descrição de hardware tradicionais, sobre um porto de saída além de poderem ser efectuadas operações de escrita, também podem ser realizadas operações de leitura. Esta alteração além de não introduzir efeitos indesejados no comportamento do circuito, simplifica a interligação de módulos, dispensando a declaração de sinais auxiliares sempre que a saída de um módulo esteja ligada tanto a entradas como a saídas de objectos nele instanciados. Na próxima secção será ilustrada uma aplicação prática desta facilidade. Um porto de saída pode ser ligado a um sinal ou a outro porto de saída ou bidireccional, mas também pode ser deixado em aberto;
- *CEaSysInOutPort* – esta classe representa um porto bidirecional (entrada/saída) e é derivada das classes

- *CEaSysInPort* e *CEaSysOutPort*. Sobre um objecto desta classe é possível efectuar operações quer de leitura quer de escrita. Um porto bidireccional tem de estar ligado a um porto do mesmo tipo ou a um sinal.

G. Núcleo de Simulação

O núcleo de simulação da linguagem EaSys é baseado em ciclos e reune um conjunto de estruturas de dados e funções responsáveis pelo controlo da simulação. A sua implementação é feita na classe *CEaSysKernel*, que disponibiliza os seguintes serviços:

- Verificação semântica do modelo do sistema;
- Detecção de eventos;
- Escalonamento das tarefas;
- Construção da hierarquia do sistema;
- Visualização dos resultados da simulação.

A classe *CEaSysKernel* não é instanciável e todos os seus métodos e atributos são estáticos para que sejam únicos dentro de cada modelo de simulação. A maior parte do métodos disponibilizados por esta classe destinam-se a ser usados pelas classes do núcleo da linguagem. Por exemplo, todos os objectos do tipo módulo, tarefa, sinal e porto são registados no núcleo de simulação durante a sua instanciação. Na versão actual da linguagem, a interacção entre a aplicação de teste de um modelo e o núcleo de simulação faz-se através dos seguintes métodos:

- `EaSys::Initialize()` – Efectua a inicialização do núcleo, devendo ser invocada pelo programa de teste logo após a instanciação do modelo a simular. Internamente, inicializa as estruturas de dados do núcleo e verifica o modelo do ponto de vista semântico;
- `EaSys::Cycle()` – Executa um dado número de ciclos de simulação. Internamente, detecta a ocorrência de eventos nos sinais, verifica quais as tarefas que devem executar e no final de cada ciclo de simulação actualiza os valores dos sinais e dos portos.

Mais à frente será descrito o processo de compilação e mostrado um exemplo de uma aplicação de teste do modelo da porta lógica complexa AOI apresentada nesta secção.

IV. EXEMPLOS

Para complementar a descrição da linguagem efectuada na secção anterior e ilustrar as potencialidades e as construções mais comuns da linguagem EaSys vão ser agora apresentadas mais algumas descrições de componentes digitais comuns. Os exemplos escolhidos para este efeito foram os seguintes:

- Porta lógica NAND de duas entradas;
- Flip-flop tipo D com entrada de inicialização assíncrona;
- Latch SR com portas lógicas NAND;
- Somador de números inteiros;
- Máquina de estados finitos.

É importante referir que a combinação das contruções e mecanismos do C++ (e.g. parametrização das classes) com as funcionalidades proporcionadas pela linguagem desenvolvida, tais como a especificação de concorrência e as abstracções de entidades existentes no hardware possibilita a construção de bibliotecas de componentes genéricos e flexíveis. Como veremos no final deste artigo, este é um dos possíveis pontos de trabalho futuro.

A. Porta lógica NAND de duas entradas

Uma porta lógica NAND de duas entradas é provavelmente um dos componentes digitais mais simples que pode ser descrito com esta linguagem. O módulo que descreve um componente deste tipo possui apenas duas entradas (in0, in1) e uma saída (out) (ver Figura 8).

O processamento é efectuado numa tarefa do tipo rotina, que é sensível a qualquer evento que ocorra numa das entradas. No corpo do método que implementa a tarefa é especificada a função booleana da porta lógica.

```
module(NAND2)
{
   iport<bool> in0;
   iport<bool> in1;
   oport<bool> out;

   method(update)
   {
      out = !(in0 && in1);
   }

   constructor(NAND2)
   {
      routine(update);
      update->sensitive(in0);
      update->sensitive(in1);
   }
};
```

Figura 8 - Descrição em EaSys de uma porta lógica NAND de 2 entradas.

B. Flip-flop tipo D com entrada de inicialização assíncrona

Um flip-flop tipo D com entrada de inicialização possui 4 portos (Figura 9):

- A entrada de dados (d);
- A entrada de sincronização (clk);
- A entrada de inicialização (clr);
- A saída de dados (q).

A actualização da saída é feita no corpo do método "*update*" que implementa uma tarefa do tipo rotina e é sensível ao flancos ascendentes (i.e. eventos positivos) da entrada de sincronização e a qualquer evento da entrada de inicialização.

```
module(FFDC)
{
   iport<bool> d;
   iport<bool> clk;
   iport<bool> clr;
   oport<bool> q;

   method(update)
   {
      if (clr)
      {
         q = false;
      }
      else if (clk.posevent)
      {
         q = d;
      }
   }

   constructor(FFDC)
   {
      routine(update);
      update->sensitive(clk, pos);
      update->sensitive(clr);
   }
};
```

Figura 9 - Descrição em EaSys de um flip-flop tipo D com entrada de inicialização assíncrona.

C. *Latch SR com portas lógicas NAND*

Nos dois exemplos apresentados acima, a operação dos módulos era descrita comportamentalmente através de tarefas e listas de sensibilidade. A latch SR é um exemplo de um módulo descrito de forma estrutural, isto é, através da instanciação e interligação de outros componentes. No caso concreto da latch SR os componentes utilizados são duas portas lógicas NAND de duas entradas cuja descrição já foi apresentada na Figura 8. A sua instanciação e interligação é efectuada no corpo do construtor (ver Figura 10). Neste caso não é necessário declarar nenhuma tarefa pois todo o processamento é realizado no interior dos componentes instanciados.

```
module(LatchSR)
{
   iport<bool> n_s;
   iport<bool> n_r;
   oport<bool> q;
   oport<bool> n_q;

   constructor(LatchSR)
   {
      instance(NAND2, gate0);
      gate0->i0(n_s);
      gate0->i1(n_q);
      gate0->o(q);

      instance(NAND2, gate1);
      gate1->i0(n_r);
      gate1->i1(q);
      gate1->o(n_q);
   }
};
```

Figura 10 - Descrição em EaSys de uma Latch SR.

D. *Somador de números inteiros*

O módulo somador da Figura 11 é semelhante à porta lógica apresentada na Figura 8, tendo sido escolhido apenas para ilustrar a realização de operações aritméticas.

```
module(Adder)
{
   iport<int> in0;
   iport<int> in1;
   oport<int> out;

   method(update)
   {
      out = in0 + in1;
   }

   constructor(Adder)
   {
      routine(update);
      update->sensitive(in0);
      update->sensitive(in1);
   }
};
```

Figura 11 - Descrição em EaSys de uma somador de quantidades inteiras.

E. *Máquina de estados finitos*

O módulo escolhido para concluir a apresentação desta sequência de exemplos de descrições foi uma máquina de estados finitos (*Finite State Machine – FSM*). Uma FSM é um modelo usado para descrever o comportamento de circuitos sequenciais e pode ser representado de forma gráfica ou textual. Apesar das suas limitações, uma das representações mais usuais é o diagrama de transição de estados (*State Transition Diagram – STD*). Na Figura 12 encontra-se o STD da FSM utilizada neste exemplo. Esta FSM segue o modelo de Moore, isto é, as saídas dependem apenas do estado actual, e possui: duas entradas (in0; in1), duas saídas (out0; out1) e quatro estados (S0; S1; S2; S3).

A forma de modelar uma FSM com a linguagem EaSys é em tudo idêntica à utilizada nas HDLs (ver Figura 13). Além da declaração dos portos, a descrição de uma FSM é constituída normalmente pelos seguintes elementos:

- Enumeração dos estados e definição do respectivo tipo. Este procedimento torna possível a declaração de sinais de um tipo específico que podem tomar apenas um conjunto restrito de valores;
- Dois sinais:
  - "*currentState*" – para armazenar o estado actual;
  - "*nextState*" – para armazenar o estado seguinte;
- Duas tarefas do tipo rotina:
  - "*sequential*" – sensível aos eventos positivos da entrada de sincronização e a qualquer evento da entrada de inicialização. Esta tarefa é responsável pelas transições de estado da FSM, implementando a memória do circuito sequencial (registo de estado);

- "*combinatorial*" – sensível a qualquer evento que ocorra nas entradas e às transições de estado. Esta tarefa implementa a componente combinatória do circuito sequencial descrito pela FSM, determinando o estado seguinte em função do estado actual e das entradas e as saídas em função do estado actual. O cálculo das saídas e do estado seguinte usa a construção "*switch-case*" do C++.

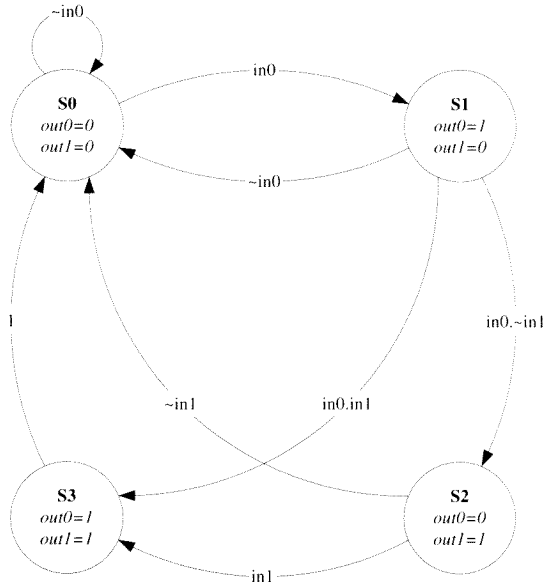


Figura 12 - Exemplo de uma máquina de estados finitos representada por um diagrama de transição de estados.

```
module(FSM)
{
   iport<bool> clk;
   iport<bool> reset;
   iport<bool> in0;
   iport<bool> in1;
   oport<bool> out0;
   oport<bool> out1;

   enum EState {S0, S1, S2, S3};

   signal<EState> currentState, nextState;

   method(sequential)
   {
      if (reset)
      {
         currentState = S0;
      }
      else if (clk.posevent)
      {
         currentState = nextState;
      }
   }

   method(combinatorial)
   {
      switch (currentState)
      {
         case S0 :
         {
            out0 = false;
            out1 = false;

            if (in0)
            {
               nextState = S1;
            }
            else
            {
               nextState = S0;
            }
            break;
         }
         case S1 :
         {
            out0 = true;
            out1 = false;

            if (!in0)
            {
               nextState = S0;
            }
            else
            {
               if (!in1)
               {
                  nextState = S2;
               }
               else
               {
                  nextState = S3;
               }
            }
            break;
         }
         case S2 :
         {
            out0 = false;
            out1 = true;

            if (in1)
            {
               nextState = S3;
            }
            else
            {
               nextState = S0;
            }
            break;
         }
         case S3 :
         {
            out0 = true;
            out1 = true;

            nextState = S0;
            break;
         }
      }
   }

   constructor(FSM)
   {
      routine(sequential);
      sequential->sensitive(clk, pos);
      sequential->sensitive(reset);

      routine(combinatorial);
      combinatorial->sensitive(currentState);
      combinatorial->sensitive(in0);
      combinatorial->sensitive(in1);
   }
};
```

Figura 13 - Descrição em EaSys de uma máquina de estados finitos.

As transições de estado podem ser escritas de uma forma mais compacta usando o operador "?:" do C++. No caso da Figura 13 poderiam ser expressas da seguinte maneira:

```
nextState = (in0 ? S1 : S0);
nextState = (in0 ? (in1 ? S3 : S2) : S0);
nextState = (in1 ? S3 : S0);
```

V. COMPILAÇÃO E SIMULAÇÃO DE UM MODELO

Os processos de compilação e simulação do modelo de um sistema escrito em EaSys são em tudo idênticos à construção e execução de uma aplicação em C++. Assim, para simular um modelo basta instanciá-lo numa aplicação de teste, compilar e executar o programa resultante. Todo o processo é efectuado com ferramentas standard de desenvolvimento em C++. O projecto deve incluir todos os ficheiros que possuem o código fonte do modelo do sistema a simular, a biblioteca que implementa a linguagem EaSys e, eventualmente, outras bibliotecas de componentes e classes utilizadas. A simulação consiste na aplicação de vectores de teste que permitam cobrir, se possível, todas as situações de funcionamento do sistema. Durante a simulação é analisada a resposta aos estímulos aplicados para assim validar o modelo elaborado para o sistema a projectar. Existem várias formas de aplicar estímulos e analisar a resposta do modelo:

- O utilizador pode aplicar manual e iterativamente os vectores de teste e observar as saídas produzidas;
- A aplicação dos vectores de teste pode ser automática, sendo as respostas produzidas armazenadas para posterior análise;
- O processo de análise dos resultados pode também ser simplificado através da detecção automática de erros e emissão de mensagens de notificação no caso de um funcionamento incorrecto ou inconsistente do modelo do sistema.

Na Figura 14 é mostrada a listagem completa do ficheiro "AOI.h" que possui o código fonte da porta lógica complexa AOI apresentada acima. Neste caso, a descrição do componente é feita num único ficheiro. Em geral, a descrição dos módulos de um sistema ou biblioteca pode ser feita de duas formas:

- Completamente em ficheiros de interface (*.h);
- Distribuída entre ficheiros de interface (*.h) e de implementação (*.cpp).

Devido à sua simplicidade, no exemplo da Figura 14 é usado o primeiro método. O segundo método possui a vantagem de permitir separar o interface da implementação e assim esconder os detalhes internos do módulo. Além das directivas para controlo de inclusão e da descrição do módulo AOI, o ficheiro "AOI.h" possui duas directivas para o preprocessador do C++ para inclusão dos ficheiros de definições da linguagem EaSys e do módulo "AND2" usado neste componente.

A listagem da Figura 15 mostra um exemplo de uma aplicação de teste em modo consola usada para simular o modelo da Figura 14. Após a instanciação e interligação do módulo "AOI" que se pretende simular é feita a inicialização do núcleo da linguagem (função `EaSys::Initialize()`). Seguidamente, a aplicação entra num ciclo infinito onde são aplicados os vectores de teste, invocado o núcleo de simulação (função `EaSys::Cycle()`) e visualizados os resultados.

```
#ifndef __AOI_H_INCLUDED__
#define __AOI_H_INCLUDED__

#include "EaSys.h"
#include "AND2.h"

module(AOI)
{
   iport<bool> in0;
   iport<bool> in1;
   iport<bool> in2;
   iport<bool> in3;
   oport<bool> out;

   signal<bool> and0Out, and1Out;

   method(nor)
   {
      out = !(and0Out || and1Out);
   }

   constructor(AOI)
   {
      instance(AND2, and0);
      and0->in0(in0);
      and0->in1(in1);
      and0->out(and0Out);

      instance(AND2, and1);
      and1->in0(in2);
      and1->in1(in3);
      and1->out(and1Out);

      routine(nor);
      nor->sensitive(and0Out);
      nor->sensitive(and1Out);
   }
};

#endif // __AOI_H_INCLUDED__
```

Figura 14 - Listagem completa do ficheiro "AOI.h" que possui a descrição da porta lógica complexa AOI.

```
#include <iostream.h>
#include <conio.h>
#include "EaSys.h"
#include "AOI.h"

int main(int argc, char* argv[])
{
   signal<bool> in0, in1, in2, in3, out;

   AOI gate("AOI");
   gate.in0(in0);
   gate.in1(in1);
   gate.in2(in2);
   gate.in3(in3);
   gate.out(out);

   EaSys::Initialize();

   while(1)
   {
      char ch = getch();

      switch (ch)
```

```
        {
            case '0' :
            {
                in0 = !in0;
                break;
            }
            case '1' :
            {
                in1 = !in1;
                break;
            }
            case '2' :
            {
                in2 = !in2;
                break;
            }
            case '3' :
            {
                in3 = !in3;
                break;
            }
            case 'q' :
            {
                exit(0);
            }
        }

        EaSys::Cycle(2);

        cout << "in3 = " << in3 << " , ";
        cout << "in2 = " << in2 << " , ";
        cout << "in1 = " << in1 << " , ";
        cout << "in0 = " << in0 << " | ";
        cout << "out = " << out << endl;
    }

    return 0;
}
```

Figura 15 - Código fonte da aplicação de teste do módulo AOI.

Após compilação e execução da aplicação podem ser aplicados estímulos de forma interactiva para avaliar a resposta do componente a vários vectores de teste. A Figura 16 ilustra os resultados obtidos para o módulo AOI ao qual foram aplicados dezasseis vectores de teste. Da análise da resposta obtida pode-se concluir que o funcionamento do componente é o correcto pois corresponde à tabela de verdade (segundo codificação de *Gray*) da respectiva função lógica.

```
C:\Tmp>AOI
in3 = 0 , in2 = 0 , in1 = 0 , in0 = 0 | out = 1
in3 = 0 , in2 = 0 , in1 = 0 , in0 = 1 | out = 1
in3 = 0 , in2 = 0 , in1 = 1 , in0 = 1 | out = 0
in3 = 0 , in2 = 0 , in1 = 1 , in0 = 0 | out = 1
in3 = 0 , in2 = 1 , in1 = 1 , in0 = 0 | out = 1
in3 = 0 , in2 = 1 , in1 = 1 , in0 = 1 | out = 0
in3 = 0 , in2 = 1 , in1 = 0 , in0 = 1 | out = 1
in3 = 0 , in2 = 1 , in1 = 0 , in0 = 0 | out = 1
in3 = 1 , in2 = 1 , in1 = 0 , in0 = 0 | out = 0
in3 = 1 , in2 = 1 , in1 = 0 , in0 = 1 | out = 0
in3 = 1 , in2 = 1 , in1 = 1 , in0 = 1 | out = 0
in3 = 1 , in2 = 1 , in1 = 1 , in0 = 0 | out = 0
in3 = 1 , in2 = 0 , in1 = 1 , in0 = 0 | out = 1
in3 = 1 , in2 = 0 , in1 = 1 , in0 = 1 | out = 0
in3 = 1 , in2 = 0 , in1 = 0 , in0 = 1 | out = 1
in3 = 1 , in2 = 0 , in1 = 0 , in0 = 0 | out = 1
```

Figura 16 - Resultados da simulação do módulo AOI com a aplicação de teste da Figura 15.

VI. COMPARAÇÃO COM O SYSTEMC

Dado que a linguagem EaSys não é a única opção disponível para utilizar o C++ como linguagem de descrição de sistemas digitais, faz todo o sentido dedicar uma secção à comparação com outra abordagem semelhante existente para o mesmo efeito. Assim, para esta comparação foi escolhida a linguagem SystemC, devido, por um lado, à sua popularidade e por outro às semelhanças com a linguagem EaSys. Em termos sintáticos e semânticos, a linguagem EaSys possui muitas semelhanças com a linguagem SystemC. Mais concretamente, os elementos básicos da linguagem são os mesmos, isto é, módulos, tarefas, sinais e portos estão disponíveis em ambos os casos, mudando apenas a forma como são declarados. Ambas as linguagens necessitam apenas de um compilador de C++ standard para construção de um modelo executável, o que relativamente a outros métodos e linguagens de especificação constitui uma vantagem para ambas. O código fonte de ambas as linguagens é aberto. No entanto, a implementação da linguagem EaSys está melhor documentada e é mais modular, o que para efeitos de extensibilidade constitui uma vantagem importante.

A linguagem EaSys tem a desvantagem de se encontrar numa fase mais primitiva do que a SystemC. No entanto, esta também pode ser uma vantagem se considerarmos que muitos dos aspectos da sintaxe e da sua implementação podem ainda, se necessário, ser facilmente alterados. Uma vantagem importante do SystemC é a sua biblioteca de sistema que permite a elaboração de modelos a níveis de abstracção mais elevados.

Na linguagem EaSys existe uma separação clara entre o núcleo de simulação, as classes que implementam as várias abstracções da linguagem e os tipos de dados. Além disso, houve a preocupação de eliminar alguma redundância existente nos tipos de dados do SystemC, reduzindo o número de tipos de dados disponíveis e aumentando a sua flexibilidade através de uma parametrização simples e intuitiva das respectivas classes. Isto permite simplificar bastante a manutenção do código de implementação da linguagem.

Uma limitação do SystemC reside no tipo de dados usados para representar as variáveis lógicas e que só podem assumir quatro valores distintos: 'Z' (alta-impedância), '0' ('zero' activo), '1' ('um' activo), 'X' (indefinido). Como veremos na próxima secção, o tipo de dados correspondente previsto para a linguagem EaSys, além dos valores anteriores suporta também os valores 'L'('zero' passivo) e 'H ('um' passivo) habitualmente disponíveis nas HDLs e úteis para a modelação de saídas passivas ligadas a barramentos.

Na linguagem EaSys foi dada uma atenção especial à definição dos operadores aplicáveis a objectos do tipo sinal e porto. O objectivo é fazer com que do ponto de vista da sintaxe a realização de uma operação de leitura/escrita sobre uma variável de qualquer tipo seja exactamente igual à realização da mesma operação sobre

um sinal ou porto com o mesmo tipo de dados. Desta forma é possível simplificar a escrita das operações porque não é necessário efectuar invocações explícitas de métodos para esse efeito. No SystemC, as operações efectuadas sobre portos devem recorrer aos métodos `read()/write()` do respectivo objecto. A Tabela 4 ilustra estas diferenças.

Operação	EaSys	SystemC

Leitura	`if (clk)`
	`{`
	`  ...`
	`}`
		`if (clk.read())`
		`{`
		`  ...`
		`}`

Escrita	`q = d;`	`q.write(d);`

Tabela 4 - Diferenças de sintaxe entre a linguagem EaSys e a SystemC nas operações de leitura e escrita em objectos do tipo porto.

O SystemC possui um problema grave na modelação de barramentos. A descrição de um barramento requer, além da declaração de um objecto que representa o meio partilhado, que os portos de saída dos módulos a ele ligados sejam de um tipo especial e diferente do utilizado nas ligações ponto a ponto. O segundo aspecto é problemático na construção de bibliotecas.

Finalmente, ao contrário do que acontece com o SystemC, a linguagem EaSys não impõe nenhuma restrição quanto ao ponto de entrada da aplicação que implementa o modelo de um sistema. Na linguagem SystemC, o ponto de entrada da aplicação deve ser uma função cujo nome é "sc_*main*", enquanto na linguagem EaSys basta que a aplicação seja compilada e que a biblioteca da linguagem seja associada durante a construção do executável. A instanciação do modelo do sistema, a inicialização do núcleo da linguagem e o controlo da simulação podem ser realizados em qualquer função, desde que executados pela ordem correcta. Apesar de ser um pormenor, este aspecto pode ser muito importante na integração de um modelo em diferentes tipos de ambientes de simulação.

VII. EXTENSIBILIDADE E TRABALHO FUTURO

Apesar da linguagem EaSys permitir já a descrição e simulação de uma gama considerável de sistemas digitais, existe ainda muito trabalho a realizar. Um dos aspectos que facilita a extensão da linguagem é a arquitectura aberta e bem documentada do seu núcleo de implementação. As possibilidades de trabalho futuro dividem-se essencialmente nos seguintes tópicos:

- Expansão das funcionalidades do núcleo da linguagem
  - implementação da classe *CEaSysThread* responsável pelos mecanismos de activação e suspensão específicos das tarefas do tipo *thread*;
  - implementação da classe *CEaSysBus* que suporta a instanciação de objectos que representam barramentos, isto é, sinais que podem ser controlados por múltiplas saídas simultaneamente. Um barramento de N bits é um sinal especial capaz de armazenar valores lógicos standard e que possui mecanismos para resolver conflitos;
- Definição de novos tipos de dados úteis para descrever hardware digital. Estes tipos deverão ser parametrizáveis para que as suas características (e.g. tamanho) possam ser definidas durante a criação dos respectivos objectos. Além disso, deve também ser implementado o subconjunto dos operadores pretendidos para cada tipo de dados. Por exemplo, no caso de uma classe que representa uma quantidade inteira, devem ser definidos os operadores aritméticos, enquanto para um vector de dígitos binários devem ser definidos os operadores lógicos. Além disso, pode também ser útil a definição de operadores de conversão entre alguns dos tipos de dados da linguagem EaSys e os existentes no C++. Os tipos de dados de interesse identificados até ao momento são:
  - bit<N> - vector parametrizável de N dígitos binários (podem possuir apenas os valores '0' ou '1');
  - logic<N> - vector parametrizável de N dígitos que representam valores lógicos standard, podendo cada um tomar um dos valores 'Z' (alta-impedância), '0' ('zero' activo), '1' ('um' activo), 'L'('zero' passivo), 'H ('um' passivo), 'X' (indefinido);
  - integer<B, N> - quantidade inteira de N bits construída com o tipo base B (char, short, long, int);
  - fixed<...> - quantidade decimal em vírgula fixa. Os parâmetros deste tipo de dados ainda não estão definidos;
- Construção de bibliotecas – tal como já foi dito atrás, a combinação das capacidades da linguagem desenvolvida com as pré-existentes no C++ permite a construção de bibliotecas de componentes genéricos e parametrizáveis. Desta forma é possível simplificar a descrição de sistemas complexos e em certos casos até sua implementação através da associação de determinados componentes da biblioteca a macro-células disponíveis na tecnologia alvo. As bibliotecas a implementar podem ser essencialmente de dois tipos:

- As que representam componentes digitais genéricos, tais como portas lógicas, registos, contadores, comparadores, (des)codificadores, (des)multiplexadores, circuitos de aritmética, memórias, ou blocos mais específicos ou complexos como filtros digitais, circuitos de interface, processadores, etc;

- As que disponibilizam construções e mecanismos úteis para modelar sistemas complexos a um nível comportamental abstracto. Um modelo que utilize estas abstracções deve ser validado e progressivamente refinado até poder ser sintetizado e implementado. Exemplos de abstracções úteis para este fim são os mecanismos de comunicação e sincronização entre módulos e tarefas, a especificação de restrições temporais, etc;

- Construção de ferramentas de síntese de hardware – dentro de determinadas restrições, uma descrição de hardware em EaSys pode ser convertida numa descrição VHDL ou Verilog ao nível RTL para posterior processamento pelas ferramentas tradicionais de síntese de hardware. Contudo, uma abordagem mais interessante e que importa considerar é a síntese directa de hardware a partir da descrição de um sistema na linguagem EaSys. Esta é provavelmente uma das tarefas mais complexas deste trabalho, mas também a mais interessante do ponto de vista de investigação;

- Criação de um ambiente de desenvolvimento integrado – para que os resultados deste trabalho sejam utilizados de uma forma fácil e intuitiva, é importante integrar as várias ferramentas construídas num ambiente de desenvolvimento. Este ambiente deve, entre outras coisas, permitir:
  - especificar e modelar o sistema a projectar;
  - automatizar a criação de programas de teste para o modelo resultante;
  - efectuar a compilação do modelo e respectivo programa de teste;
  - correr a especificação executável do sistema para efeitos de simulação;
  - visualizar intuitivamente os resultados obtidos;
  - refinar de forma sistemática o modelo até que este possa ser sintetizado;
  - sintetizar o hardware que implementa o sistema;
  - invocar as ferramentas de implementação específicas da tecnologia utilizada;

- Análise do desempenho e da quàlidade dos resultados produzidos (*benchmarking*) – para qualquer linguagem e ferramenta de projecto é fundamental comparar os tempos de execução e a qualidade final dos resultados obtidos com outras técnicas e ferramentas disponíveis para o mesmo fim;

- Adaptação do núcleo da linguagem EaSys a outras linguagens base - a biblioteca de classes que implementa a linguagem EaSys pode ser reescrita noutras linguagens orientadas por objectos, sendo a linguagem JAVA uma das possibilidades mais interessantes devido à sua crescente utilização quer no desenvolvimento do software, quer como linguagem de especificação.

VIII. CONCLUSÕES

A utilização de linguagens de programação tradicionais nas fases iniciais do projecto de sistemas complexos para efeitos de especificação e validação é uma abordagem cada vez mais utilizada. Por outro lado, a uniformização das linguagens usadas ao longo de todas as etapas de projecto possui vantagens importantes ao nível da produtividade e consequentemente no tempo de desenvolvimento. A linguagem EaSys apresentada neste artigo adiciona à linguagem de programação orientada por objectos C++ um conjunto de funcionalidades e mecanismos que a tornam adequada também para descrever hardware. A abordagem utilizada para estender a linguagem C++ consiste numa biblioteca de classes permitindo a utilização de ferramentas standard de desenvolvimento em C++ para descrever, compilar e simular um modelo de um sistema. Esta linguagem proporciona uma plataforma uniforme para a modelação, simulação, verificação e síntese de sistemas complexos compostos tanto por componentes de hardware como de software. A arquitectura do núcleo de implementação da linguagem é extremamente compacto, aberto, modular, flexível, extensível e portável podendo ser utilizado em qualquer plataforma que possua um compilador de C++ standard. A disponibilização do código fonte e das bibliotecas pré-compiladas para as plataformas Microsoft Windows 2000 e Linux está prevista para o início do quarto trimestre de 2001.

REFERÊNCIAS

[1] D. Verkest, J. Kunkel and F. Schirrmeister, "System Design using C++", *Proceedings of DATE-2000*, pp. 74-83, Paris, France, March 2000.

[2] R. Helaihel and K. Olukotun, "JAVA as a Specification Language for Hardware/Software Systems", *Proceedings of ICCAD-97*, pp. 690-697, San Jose, CA, November 1997.

[3] P. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann, 1996.

[4] D. Thomas and P. Moorby, "The Verilog Hardware Description Language", Kluwer Academic Publishers, 1996.

[5] R. Gupta and S. Liao, "Using a Programming Language for Digital System Design", *IEEE Design and Test of Computers*, pp. 72-80, April-June 1997.

[6] A. Ghosh, J. Kunkel, and S. Liao, "Hardware Synthesis from C/C++", *Proceedings of DATE-99*, pp. 387-389, Munich, Germany, March 1999.

[7] HandelC, http://www.celoxica.com/.

[8] Open SystemC Initiative, http://www.SystemC.org/ .

[9] OCAPI, http://www.imec.be/ocapi/.

[10] Cynlib, http://www.CynApps.com/.

[11] B. Stroustrup, "The C++ Programming Language", 2nd Edition, Addison-Wesley Publishing Company, 1995.

Especificação e Simulação Interactiva de Algoritmos de Controlo Paralelos e Hierárquicos*

Andreia Melo, Valery Sklyarov, António Ferrari

Resumo – **O método de especificação apresentado neste artigo é utilizado para descrever o comportamento de uma unidade de controlo digital. Os algoritmos de controlo que gerem esta unidade seguem um modelo hierárquico, paralelo ou uma conjugação de ambos. Neste artigo é abordada a especificação e a simulação dos algoritmos de controlo paralelos e hierárquicos em simultâneo. O estudo de caso apresentado neste artigo consta da especificação da unidade de controlo do controlador CAN – *Controller Area Network* utilizando os HiParaGraphs.**

Abstract – **The specification method, which is presented in this paper, is used to describe the behavior of a digital control unit. The control algorithms that manage this unit may be hierarchical, parallel and both. In this paper is shown the specification and simulation of hierarchical and parallel control algorithms at the same time. The CAN - *Controller Area Network* protocol was used as a case study being specified using HiParaGraphs.**

I. INTRODUÇÃO

Existem sistemas digitais que se podem dividir em duas unidades, a unidade de controlo e a unidade de execução. A primeira, que vai ser abordada neste artigo, tem como função estabelecer a sequência de operações que deverão ser efectuadas pela unidade de execução. Assim, é necessário definir um algoritmo de controlo para decidir a sequência de operações. Dependendo da complexidade da unidade de controlo em causa, podemos utilizar vários tipos de algoritmos. O caso mais simples é a existência de um único algoritmo que estabelece uma sequência de operações, baseado ou não em valores lógicos de sinais de entrada, e por isso com características reactivas, actuando em sinais de saída binários. No entanto, a complexidade da unidade de controlo pode ser tal que a implementação de apenas um algoritmo não seja tão eficiente como a sua divisão em sub-algoritmos mais simples. Desta forma, temos um conjunto de sub-algoritmos que podem ser interligados de várias maneiras. De forma hierárquica, isto é, existe um conjunto de sub-algoritmos cujas invocações estão originadas segundo uma árvore. No topo da árvore encontra-se aquele a que chamamos sub-algoritmo principal porque tem como função iniciar a execução do algoritmo de controlo. Numa hierarquia um sub-algoritmo pode invocar outro, e assim sucessivamente, até que todos terminem a sua execução, voltando aos seus invocadores, isto é, percorrendo o caminho inverso. Por outro lado, podemos ter um conjunto de sub-algoritmos que pretendamos executar em paralelo. Neste caso não existe um algoritmo principal mas um conjunto de sub-algoritmos onde dois ou mais são executados em simultâneo. Por fim, o caso que vamos aqui abordar é um conjunto de sub-algoritmos cuja interacção é hieráquica e ao mesmo tempo paralela, ou seja, existe um algoritmo principal onde podem ser invocados vários sub-algoritmos simultaneamente e estes, por sua vez, também podem ter este comportamento. A especificação de todos estes tipos de algoritmos é efectuada recorrendo aos Grafos Hierárquicos e Paralelos – HiParaGraphs (***Hierarchical & Parallel Graphs***) [1, 2]. Um HiParaGraph descreve algoritmicamente o comportamento de unidades de controlo digitais, sendo composto por diferentes tipos de nodos ligados entre si, por intermédio de arcos direccionados, que estabelecem o fluxo de execução.

A simulação de um algoritmo complexo, hierárquico e paralelo será aqui abordada. Foi efectuada no mesmo ambiente de desenvolvimento da própria especificação, ou seja no GraphBuilder [3, 4], e pode ser utilizada para efeitos de depuração. Durante este processo é possibilitada a intervenção do utilizador em qualquer instante. O modelo de implementação que suporta estes circuitos de controlo encontra-se ainda em desenvolvimento. No entanto, o método utilizado na simulação sugere uma arquitectura.

A especificação da unidade de controlo do controlador CAN – *Controller Area Network* [5, 6], é aqui apresentada como um estudo de caso com o objectivo de evoluir a linguagem aqui apresentada.

Este artigo está dividido em cinco secções. A primeira corresponde à presente introdução. Na segunda secção descrevem-se as características da linguagem de especificação responsáveis pela interacção entre os vários sub-algoritmos. A secção três é dedicada à simulação do algoritmo de controlo com possibilidade de interacção

* Este trabalho foi financiado pela bolsa de doutoramento da Fundação para a Ciência e Tecnologia SFRH/BD/971/2000.

com o utilizador. Um estudo de caso é apresentado na secção quatro. A secção cinco contém as conclusões.

II. A LINGUAGEM DE ESPECIFICAÇÃO

Os HiParaGraphs são uma linguagem de especificação gráfica formal de algoritmos de controlo que se baseiam no formalismo da máquina de estados finitos (FSM – *Finite State Machine*). Possuem as propriedades sequenciais e simples dos diagramas de transição de estados e combinam a hierarquia, modularidade e comunicação dos Statecharts [7] com o paralelismo das redes de Petri [8] e dos Algoritmos de Controlo Lógicos [9], utilizando uma notação gráfica e intuitiva.

Os vários módulos de um HiParaGraph possuem características diferentes e por isso foram divididos em dois tipos: **funções** e **procedimentos** [1]. As funções são sempre responsáveis por invocações hierárquicas porque o sub-algoritmo que as invoca necessita de esperar pela sua terminação. Quando esta necessidade de espera não acontece, geralmente no caso dos procedimentos, estes são invocados em paralelo, pois neste caso mais do que um sub-algoritmo está activo ao mesmo tempo.

Nas próximas sub-secções são apresentadas, ao nível comportamental, algumas características da especificação de uma unidade de controlo. Para as ilustrar são utilizados grafos dirigidos [10] onde cada nodo representa um sub-algoritmo e cada arco dirigido corresponde a uma possível invocação do sub-algoritmo a que se destina. Os retornos não são aqui representados por uma questão de simplicidade de representação. Dentro de cada nodo existe um identificador do sub-algoritmo, que consiste num número inteiro, para que se possam distinguir as várias invocações, até porque estas podem ocorrer de forma repetida. Os vários nodos são coloridos com cores diferentes de forma a representar possíveis caminhos de invocações e quais os sub-algoritmos activos num dado instante da execução do algoritmo de controlo.

A. A especificação de hierarquia

A especificação hierárquica de um algoritmo de controlo justifica-se quando o comportamento de uma unidade de controlo é descrito por um algoritmo extenso, de tal forma que se poderia dividir em módulos, facilitando tanto a compreensão da especificação como a implementação do respectivo circuito. Um algoritmo de controlo quando especificado hierarquicamente possui um sub-algoritmo que invoca outro e este, por sua vez, será executado num nível hierárquico inferior. Quando este terminar a sua execução retorna ao algoritmo que o invocou, ou seja, ao nível imediatamente superior. Na fig. 1 é apresentado um grafo dirigido dividido em tres partes pelas linhas a tracejado. Cada parte corresponde a um nível da hierarquia do algoritmo de controlo que este grafo pretende representar. Tal como já foi introduzido anteriormente, cada nodo representa um sub-algoritmo e como podemos ver na fig. 1 os nodos estão coloridos com cores diferentes. Aqui está ilustrado um exemplo que mostra um caminho de execução do algoritmo. A branco estão aqueles sub-algoritmos que não foram invocados. A cinzento claro estão indicados os sub-algoritmos que foram invocados para que seja fácil distinguir na figura o caminho percorrido, 1→3→5. Como se pode verificar, existe apenas um sub-algoritmo colorido em cada nível hierárquico. No último nível hierárquico, a cinzento escuro está representado o sub-algotimo que está a ser executado (5). Quando este terminar o fluxo do algoritmo corresponderá ao caminho inverso (retorno), 5→3→1.

Os HiParaGraphs permitem a especificação da hierarquia quando prevêm a invocação de funções, pois o sub-algoritmo que as invoca precisa de esperar pelo resultado por elas calculado. Só após a determinação deste resultado é que pode prosseguir pois este decidirá o caminho após o nodo que especifica a invocação da função.

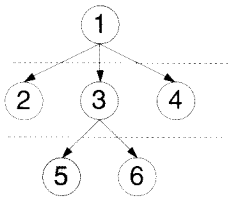


Figura 1 Grafo de representação de hierarquia.

B. A recursividade

A rede de invocações ou a interacção entre os vários sub-algoritmos de um HiParaGraph pode ser tal que além de invocarem sub-algoritmos diferentes podem também invocar-se a eles próprios. Assim, este método permite a especificação de algoritmos de controlo recursivos. Tal como já foi mencionado em [1] a recursividade infinita pode ser detectada mediante a construção de um HiParaGraph equivalente. A correcta execução do algoritmo de controlo depende do modelo de implementação escolhido e dos recursos de hardware disponíveis. Os modelos de implementação que utilizam memórias de *stack*, que embora limitem a recursividade do algoritmo devido ao tamanho máximo da memória, (podendo ocorrer uma situação de *stack overflow*) são aqueles que melhor se adequam a este tipo de algoritmos. O conhecimento prévio da profundidade do *stack* e a construção do HiParaGraph equivalente, representando a árvore de invocações, poderá evitar esta situação.

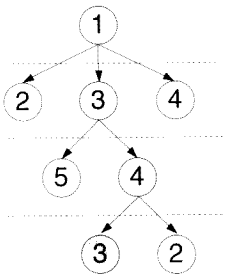


Fig. 2 – Grafo de representação de recursividade numa especificação hierárquica.

Na fig. 2 o grafo representa uma situação de recursividade porque o caminho percorrido pelo algoritmo de controlo ao longo da hierarquia é 1→3→4→3. Como podemos concluir, o sub-algoritmo identificado com o número 3 é invocado pela segunda vez, sendo assim este algoritmo é denominado recursivo pois estamos perante um ciclo fechado de invocações: 1→3→4→3→4→3→... Este ciclo termina apenas quando se verificar uma condição tal que 1→3→4→3→4→3→...→3→4→2.

De notar que a condição necessária a um algoritmo de controlo para ser recursivo é que este seja hierárquico.

Se considerarmos a recursividade existente na fig. 2, vemos que apenas uma instância do sub-algoritmo 3 está activa num dado instante. No entanto, se este grafo fosse alterado como mostra a fig. 3, temos um sub-algoritmo que se invoca a ele próprio (3). Neste caso além de recursividade existe também reentrância pois o sub-algoritmo 3 é iniciado novamente antes da sua terminação. Para evitar a recursividade infinita, isto é, para que este ciclo de invocações não se feche definitivamente, deve existir sempre uma condição que permita a terminação do sub-algoritmo em questão antes que ele se invoque novamente a ele próprio ou uma invocação a um sub-algoritmo diferente (2) que já irá mudar o fluxo do algoritmo de controlo, tornando possível efectuar o caminho de retorno, quebrando a recursividade.

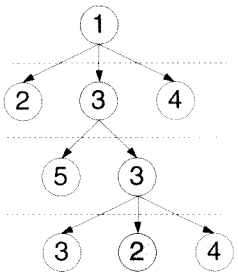


Figura 3. Grafo de representação simultânea de recursividade e reentrância.

C. *A especificação do paralelismo*

A especificação paralela é útil nos casos em que se pretende executar duas ou mais tarefas em simultâneo, como mostra o grafo da fig. 4. Todos os nodos estão coloridos a cinzento escuro porque estão activos, ou seja, a executar simultaneamente.

Figura 4. Grafo de representação de paralelismo.

Os HiParaGraphs suportam o paralelismo especificando a invocação de dois ou mais procedimentos dentro do mesmo nodo de activação. Neste caso, como irá ser detalhado mais adiante, os HiParaGraphs dispõem de indicadores de activação que diferenciam as várias formas de invocar sub-algoritmos dentro do mesmo nodo.

No entanto, existe outra forma de invocação, que será mencionada numa das secções posteriores, em que um sub-algoritmo pode ser invocado paralelamente aos restantes que já estão a ser executados, por intermédio de outro tipo de nodos integrados em sub-algoritmos diferentes.

D. *A especificação simultânea de hierarquia e paralelismo*

Um dos objectivos dos HiParaGraphs é a especificação simultânea da hierarquia e do paralelismo, isto é, uma combinação das situações apresentadas nas sub-secções anteriores. Para ilustrar este caso usamos o exemplo da fig. 5 onde se representa um algoritmo com quatro níveis hierárquicos. A principal diferença deste grafo relativamente aos anteriores é a activação de mais de um nodo por nível hierárquico. Os HiParaGraphs são capazes de invocar mais do que um sub-algoritmo ao mesmo tempo [1] e por isso aumentamos assim o número de nodos coloridos, como vemos na fig. 5. Existem quatro níveis hierárquicos e a execução foi iniciada em 1. Este invocou simultaneamente os sub-algoritmos 3 e 4, e o sub-algoritmo 3 invocou o 2 e o 5. Num dado instante, que mostra a figura, estão activos três sub-algoritmos em dois níveis hierárquicos diferentes.

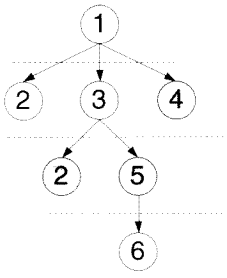


Figura 5. Grafo de representação simultânea de hierarquia e paralelismo.

De notar que na fig. 5 todos os sub-algoritmos activos são diferentes (4, 2 e 5). No entanto, e tal como já foi abordado anteriormente, (ver fig. 3) pode acontecer a ocorrência de uma invocação de um sub-algoritmo que já esteja a ser executado. A esta situação chamamos reentrância e será explicada na próxima sub-secção. Um algoritmo de controlo pode ser recursivo e reentrante simultaneamente quando um dos seus sub-algoritmos se invoca a ele próprio.

E. *A reentrância*

Os algoritmos hierárquicos e paralelos possuem mais do que um sub-algoritmo activo simultaneamente. Numa invocação, quer seja hierárquica quer paralela, o algoritmo de controlo poderá especificar a invocação de um sub-algoritmo que já esteja a ser executado. A este caso chama-se reentrância pois teremos de iniciar/entrar num sub-algoritmo já em execução.

Na fig. 6 mostra-se um grafo com quatro níveis hierárquicos e onde podemos ver que três sub-algoritmos estão activos simultaneamente, pois existem três nodos coloridos a cinzento escuro. No entanto, dois destes possuem a mesma identificação (3). Neste caso terá de ser criada uma nova instância do sub-algoritmo invocado repetidamente.

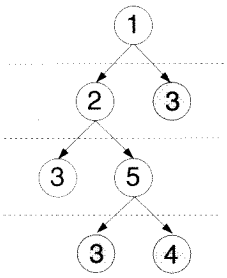


Figura 6. Grafo de representação de reentrância numa especificação hierárquica e paralela.

Os HiParaGraphs prevêm esta situação, que será abordada nas secções posteriores. Durante a simulação de um algoritmo de controlo com estas características, a ferramenta de simulação permite ao utilizador que disponha ou não de reentrância, isto é, esta é uma propriedade apresentada como opção pois o utilizador pode estar ou não limitado ao nível da implementação do respectivo circuito de controlo.

F. Os nodos utilizados em invocações de sub-algoritmos

Considerando um algoritmo de controlo complexo constituído por um conjunto de sub-algoritmos, são necessários nodos com capacidades de invocação e sincronismo que assegurem o correcto funcionamento da unidade de controlo, sendo esta especificada por um algoritmo paralelo e hierárquico. Tal como foi apresentado em [1], um HiParaGraph possui quatro categorias de nodos iniciação, terminação, activação e teste. No entanto, só os dois últimos serão aqui detalhados.

Os sinais de saída da unidade de controlo são os responsáveis pela produção da sequência de operações na unidade de execução. Estas operações (elementares), são chamadas microoperações. $Y=\{y_1,\ldots,y_n\}$ é o conjunto de microoperações induzidas pelos sinais binários $y_1,\ldots,y_n$ da unidade de controlo. Para activar a microoperação y_n ($n=1,\ldots N$) o sinal $y_n=1$ deverá ser activado. Por vezes algumas microoperações são executadas em simultâneo (no mesmo ciclo de relógio) na unidade de execução, que é equivalente à introdução de mais do que uma microoperação dentro de um único nodo.

Os procedimentos, por vezes também denominados macrooperações, pertencem ao conjunto $Z=\{z_1, z_2, \ldots, z_N\}$. Os nodos de activação dos HiParaGraphs podem simultaneamente especificar a activação de sinais de saída e a invocação de sub-algoritmos, ou seja, especificar elemento(s) do conjunto Y e elemento(s) do conjunto Z dentro do mesmo nodo. A cada sinal pode ser atribuído um indicador de activação, individualmente, cuja função será detalhada na secção seguinte.

Os nodos do tipo *Sync* são também utilizados para invocar procedimentos, mas neste caso de forma indirecta. Nestes nodos são colocados sinais pertencentes ao conjunto $S=\{s_1, s_2, \ldots, s_M\}$ e que são utilizados na comunicação entre sub-algoritmos. Foram criados para que seja possível invocar um procedimento ou função quando se atingem determinados estados de sub-algoritmos diferentes. Para tal é também necessário especificar parâmetros de entrada nos sub-algoritmos invocados desta forma.

A sequência de activação das microinstruções é determinada por funções de transição, isto é, funções Booleanas α_{ij} ($i,j=1,\ldots T$) de variáveis Booleanas $X=\{x_1,\ldots x_L\}$. Estas variáveis correspondem às variáveis de entrada da unidade de controlo, também denominadas condições lógicas, e podem ser alteradas pelas microoperações, devido à possibilidade de existência de feedback nas unidades de controlo.

Para testar os valores lógicos destes sinais de entrada são usados os nodos de teste. A invocação de funções Booleanas, elementos do conjunto $F=\{f_1, f_2, \ldots, f_J\}$ é efectuada dentro dentro destes nodos e pode influenciar o fluxo de execução do algoritmo invocador a partir de um valor que é determinado e devolvido pela função, decidindo assim qual o próximo estado activo do circuito. Cada nodo invoca apenas uma função lógica, isto é, especifica apenas um elemento do conjunto F. O valor binário calculado pela função é devolvido utilizando um nodo do tipo *Assign*. Existem apenas dois nodos deste tipo dentro da especificação da função lógica, um especificando o valor lógico 0 e outro especificando o valor lógico 1. No entanto, se quisermos considerar funções f mais complexas, que devolvem outros valores além do 0 e do 1, deverá ser utilizado outro tipo de nodo e dentro do sub-algoritmo que especifica a função deverão existir tantos nodos *Assign* como todos os valores possíveis tomados pela função. Futuramente os HiParaGraphs incluirão esta forma de especificar funções com vários valores de retorno.

G. Os indicadores de activação em sinais de saída

Dentro dos nodos de activação podem ser utilizados indicadores de activação, quer em sinais de saída quer em invocações de procedimentos. Estes indicadores são atribuídos individualmente a cada sinal. No caso dos sinais de saída temos os indicadores SET – "S" e RESET – "R" de forma a ser possível colocar um sinal binário a 1 ou 0, respectivamente, durante mais do que um ciclo de relógio. Caso se pretenda efectuar estas operações apenas durante o ciclo de relógio em que o nodo está activo podem usar-se as opções de atribuição dos valores lógicos 0 ou 1 a cada sinal de saída. Estes indicadores já foram

apresentados em [1] e o método utilizado na sua implementação detalhado em [3].

H. As diferentes formas de invocação de sub-algoritmos

De forma a distinguir as diferentes formas de invocação de sub-algoritmos, neste caso vamos considerar os procedimentos, os HiParaGraphs dispõem de indicadores de activação associados a cada procedimento e que podem ser os seguintes:

- NONE - nenhum
- WAIT - espera
- RESET - reiniciação
- STOP - terminação

Dentro do mesmo nodo de activação existem sempre sub-conjuntos de Z que estão afectados por estes indicadores, mesmo que sejam sub-conjuntos vazios.

Estes indicadores podem ser divididos em duas categorias. Enquanto NONE – " " e WAIT – "W" estão associados à invocação dos procedimentos, RESET – "R" e STOP – "S" estão indicados à intervenção sobre um dado procedimento que já está a ser executado. De seguida serão detalhadas as funções de cada indicador.

Um procedimento que seja especificado com NONE dentro de um dado sub-algoritmo significa que será iniciado e o sub-algoritmo que o invocou continuará a sua execução. Por outro lado, se for especificado com WAIT o sub-algoritmo invocador ficará à espera da terminação daquele que invocou para poder prosseguir. O indicador RESET fará com que o procedimento especificado, caso este esteja a ser executado, reinicie a sua execução, ou seja, activará imediatamente o seu nodo Begin. Contrariamente a este o indicador STOP terminará o procedimento especificado, caso esteja a ser executado, activando o respectivo nodo End.

Dentro do mesmo nodo de activação podem existir todas as combinações possíveis de sinais com indicadores de activação diferentes. Isto significa que poderá acontecer que o algoritmo invocador poderá especificar, por exemplo, procedimentos com os indicadores NONE e WAIT. Neste caso deve invocar todos eles e ficar apenas à espera da terminação daqueles especificados com WAIT.

III. A SIMULAÇÃO

GraphBuilder é a ferramenta de software que suporta esta linguagem de especificação. A sua constante evolução relativamente às versões anteriores [7, 8] consistiu na integração da hierarquia com o paralelismo ao nível da especificação e a possibilidade de simulação do algoritmo de controlo paralelo e hierárquico utilizando o mesmo ambiente de desenvolvimento.

I. O algoritmo de simulação

A simulação de um algoritmo de controlo paralelo e hierárquico é feita passo a passo, calculando ou determinando cada estado seguinte de cada sub-algoritmo que está a ser executado. Num HiParaGraph todos os sub-algoritmos que o constituem podem ser executados simultaneamente, assim como várias instâncias do mesmo sub-algoritmo nos casos de recursividade. A forma como é calculado o estado seguinte de cada um não pode depender da ordem pela qual os algoritmos são percorridos pois este cálculo não deverá ser influenciado pelos restantes.

A existência de um modelo de memória *stack* revelou-se fundamental durante este processo de forma a permitir a simulação de hierarquia e recursividade, pois nestes casos é necessário o conhecimento de alguns estados anteriores do algoritmo de controlo para que possam retornar ao seu invocador e ao estado em que este foi interrompido. No entanto, além da hierarquia e da recursividade temos o paralelismo e para tal o modelo de memória baseou-se num *stack* bidimensional onde a profundidade corresponde ao número de invocações hierárquicas de funções e a largura ao número de invocações paralelas de procedimentos. A fig. 7 pretende ilustrar uma possível forma do *stack* bidimensional durante uma simulação.

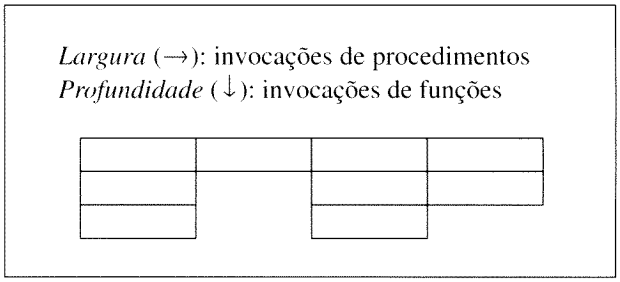


Figura 7. Possível estado do stack bidimensional durante uma simulação.

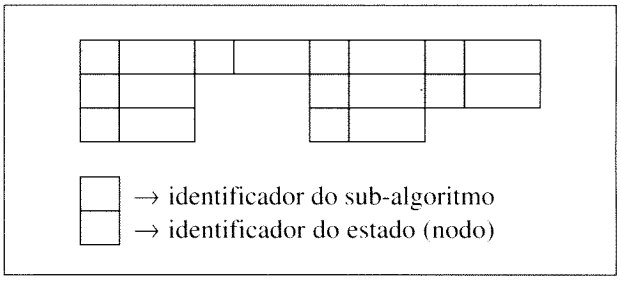


Figura 8. Os diferentes campos de cada registo do stack bidimensional.

Cada quadrícula da memória de *stack* repesentada na fig. 7 representa um registo. No entanto, e tal como faz parte do objectivo dos HiParaGraphs, de forma a flexibilizar o comportamento do algoritmo de controlo, cada registo é dividido em dois campos (ver fig. 8). O primeiro contém o identificador do sub-algoritmo e o segundo o identificador do estado dentro desse sub-algoritmo. Esta é uma forma de codificar os estados do algoritmo de controlo que se pretende que seja mais eficiente do que a codificação binária de todos os estados. Esta eficiência e flexibilidade revela-se ao nível da utilização dos indicadores de activação, como por

exemplo RESET e STOP, onde é necessário efectuar a procura de um ou mais sub-algoritmos que já estejam a executar, ou seja, que já estejam presentes no *stack*. Para identificar um sub-algoritmo nestas condições basta analisar o identificador do sub-algoritmo do último registo de cada coluna do *stack* bidimensional.

Tal como já foi mencionado anteriormente, a ordem pela qual são determinados os estados seguintes de cada sub-algoritmo não deve ser tida em conta porque teoricamente todos os procedimentos são executados ao mesmo tempo e embora possa haver interacção entre eles, o cálculo do novo estado de um dado sub-algoritmo não deve influenciar o cálculo de um novo estado de outro que esteja a ser executado em paralelo. Assim, na simulação efectuada em software foi utilizada uma estrutura de dados adicional, que consiste num conjunto de registos, para guardar o estado anterior da simulação. O seu número de registos é igual ao número de registos da largura do *stack* bidimensional. Por exemplo, no caso da fig. 8 esta estrutura possui os últimos quatro registos do *stack* bidimensional.

Assim, cada passo da simulação pode ser dividido em três etapas:

1. A estrutura de salvaguarda é apagada e é criada uma nova com a mesma largura do *stack* bidimensional, para onde os estados actuais do *stack* são copiados.
2. Todas as colunas desta estrutura são percorridas e aqui são guardados os estados seguintes de cada sub-algoritmo.
3. Os novos valores contidos nesta estrutura são copiados para o *stack* bidimensional.

Em cada ciclo de relógio, ou ciclo de simulação, existe a possibilidade de invocar sub-algoritmos e/ou de estes retornarem ao seu invocador. Neste casos o *stack* bidimensional pode aumentar ou diminuir de tamanho em ambas as dimensões. Para tal, a estrutura de salvaguarda que é utilizada é composta por três campos de identificadores: o do sub-algoritmo, o do estado dentro desse sub-algoritmo e o da operação que deve ser efectuada *na coluna correspondente do stack bidimensional*. As operações a efectuar são pré-definidas e identificadas da seguinte maneira:

- NOOP – Nenhuma operação (utilizada na inicialização).
- PUSH – Adição de um novo registo de estado.
- POP – Remoção do último registo de estado.
- REFRESH – Alteração dos valores existentes do último registo de estado.
- REMOVE – Remoção de uma das colunas do *stack* bidimensional.
- FLUSH – Apaga todo o conteúdo da respectiva coluna do *stack*, embora não o remova.
- FLUSH_PUSH – Apaga todo o conteúdo da respectiva coluna do *stack* e é utilizado apenas na terminação do sub-algoritmo principal. De notar que esta operação corresponde à junção de duas pois as duas operações de FLUSH e PUSH, neste caso, devem ser efectuadas no mesmo ciclo de relógio.

Na fig. 9 é apresentada a simulação de um algoritmo de controlo constituído por cinco sub-algoritmos. Existe um algoritmo principal, isto é, aquele que é inicialmente executado, duas funções e dois procedimentos. De notar que este algoritmo foi construído e escolhido apenas como exemplo para a simulação simultânea da hierarquia e paralelismo.

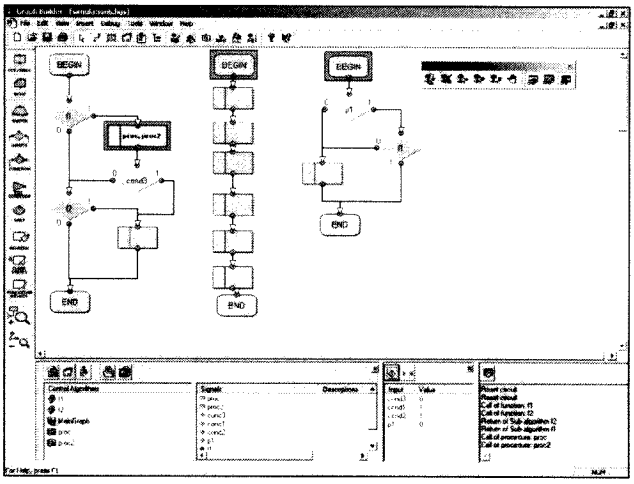


Figura 9. Simulação de um HiParaGraph utilizando o GraphBuilder.

Neste caso são visualizados três sub-algoritmos a executar em paralelo. De notar que os nodos de activação surgem vazios, ou seja, não activam nenhum sinal, porque para este exemplo a activação dos sinais de saída da unidade de controlo é um factor irrelevante. O que neste artigo se pretende evidenciar é apenas a interacção entre os vários sub-algoritmos de um HiParaGraph. À esquerda está representado o sub-algoritmo principal que possui um nodo activo, pois está realçado relativamente aos restantes. Este é um nodo de activação que especifica a invocação (com indicador NONE) de dois procedimentos. Estes, por sua vez, foram iniciados e por isso são visualizados à direita do sub-algoritmo invocador, onde podemos ver os nodos Begin activos de cada um deles. As janelas situadas na parte inferior da aplicação, são utilizadas para visualizar os sinais de entrada e o *log* da simulação, onde são fornecidas mensagens relativas às acções que estão a ser efectuadas pelo algoritmo de controlo. Na fig. 10 estas janelas estão representadas mais pormenorizadamente. À semelhança da visualização das entradas existe também uma outra janela que mostra todos os sinais de saída e os respecivos valores binários. No entanto, esta não está aqui presente porque tal como já se disse anteriormente, neste exemplo não foram utilizados sinais de saída pois eram irrelevantes para a demonstração do processo de simulação. Estas janelas são flutuantes, no entanto podem ser encostadas à parte inferior da janela principal, e possuem uma barra de ferramentas com botões para actuar nos valores binários dos sinais de entrada e fazer o seu reset, no caso da janela de entradas. O botão

existente no topo da janela de *log* serve apenas para apagar o seu conteúdo.

Na fig. 11, à esquerda, está representado o sub-algoritmo principal que no primeiro ciclo de simulação invoca a função f1. O momento da simulação mostrado nesta figura corresponde ao passo da simulação em que se consideram os valores lógicos dos sinais de entrada cond3 = 0 e p1 = 1, depois de f1 ter retornado com o valor lógico 1.

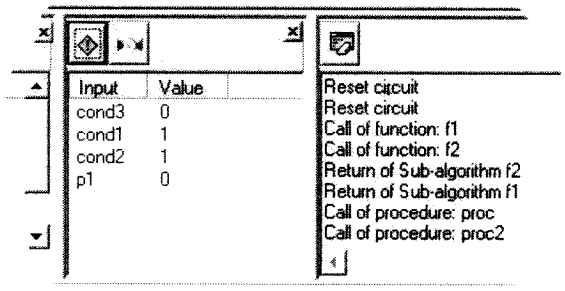


Figura 10. Pormenor da visualização dos valores das entradas e das mensagens de log.

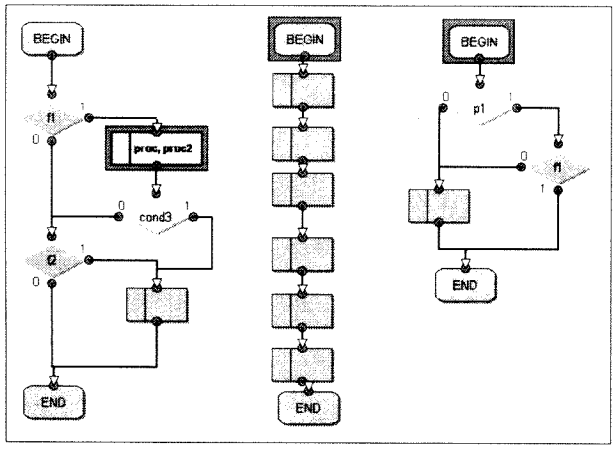


Figura 11. Invocação de dois procedimentos em paralelo.

Neste momento o estado do *stack* aumentou a sua largura de 1 (o sub-algoritmo principal tinha apenas invocado hierarquicamente a função f1) para 3, pois agora temos três sub-algoritmos a executar simultaneamente. Na fig. 12 é mostrado o estado do *stack* com os respectivos identificadores dentro de cada registo.

0	4	1	0	2	0

Figura 12. Estado do stack bidimensional relativamente ao estado da simulação da fig. 11.

Assim, no próximo ciclo de relógio teremos uma transição hierárquica do sub-algoritmo da esquerda porque como cond3 = 0 a função f2 deverá ser invocada. O segundo sub-algoritmo efectua uma simples transição de estado e por fim o terceiro, como p1 = 1, invoca f1, dando-se assim uma nova transição hierárquica. Como se pode ver na fig. 13, os nodos Begin das funções f2 (à esquerda) e f1 (à direita) encontram-se activos.

Assim, da esquerda para a direita temos os seguintes sub-algoritmos a executar: f2, proc e f1. Neste momento temos duas hipóteses a considerar. Se o sinal de entrada cond1, como se mostra na fig. 12 for igual a 0, então o nodo *Assign* de valor 0 ficará activo. Por outro lado, e esta é a situação que vamos considerar, se cond1 = 1 então será necessário invocar novamente a função f2. No entanto, esta já está a ser executada (sub-algoritmo à esquerda) e por isso estamos perante uma situação de reentrância. Nesta aplicação é possível escolher entre executar ou não um algoritmo reentrante, isto é, se a reentrância for uma propriedade aceite pelo utilizador então f2 deverá ser invocada e visualizada à direita. Caso contrário, o sub-algoritmo f1 terá de esperar pela terminação da instância de f2, que já está a ser executada, para poder prosseguir.

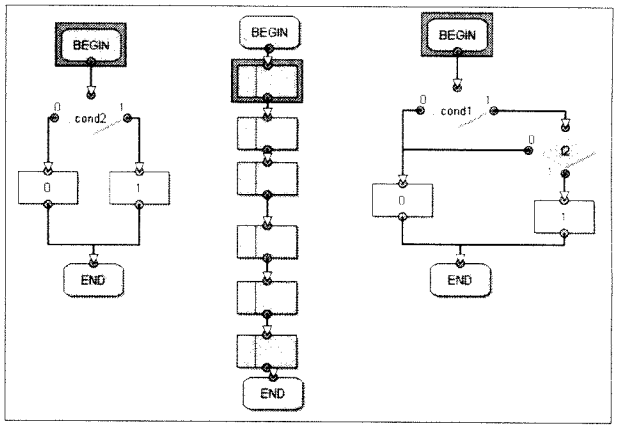


Figura 13. Invocações hierárquicas de sub-algoritmos que executam em paralelo, relativamente à fig. 11.

Na fig. 14 podemos ver pelo estado do *stack* as duas invocações hierárquicas (à esquerda e à direita) que ocorreram. O procedimento representado no centro da fig. 13 continua a sua execução e por isso o registo do *stack* na coluna a que lhe corresponde (ao centro) foi apenas actualizado com o identificador do novo estado activo.

0	3	1	2	2	3
4	0			3	0

Figura 14. Estado do stack bidimensional relativamente ao estado da simulação da fig. 13.

Esta é a situação que é apresentada na fig. 15. f1 permanece com o seu nodo Begin activo, à espera da terminação de f2, enquanto os restantes transitam de estado.

Três ciclos de relógio mais tarde, situação a que corresponde a fig. 16, a função f2 já retornou ao algoritmo invocador e por isso f1 já pode efectuar a invocação de f2 pela qual esperava. Nesta figura, da esquerda para a direita já podemos visualizar os sub-algoritmos: principal, proc e f2, com o respectivo nodo Begin activo.

Dois ciclos depois do estado da simulação representado na fig. 16, a função f2 atinge o seu nodo End ao mesmo tempo que no procedimento proc também o nodo End é activado. O algoritmo principal reinicia a sua execução, activando o respectivo nodo Begin. Esta situação está representada na fig. 17 e o correspondente estado do *stack* na fig. 18. Assim, ambos os sub-algoritmos proc e f2 vão terminar e por isso deixarão de ser visualizados.

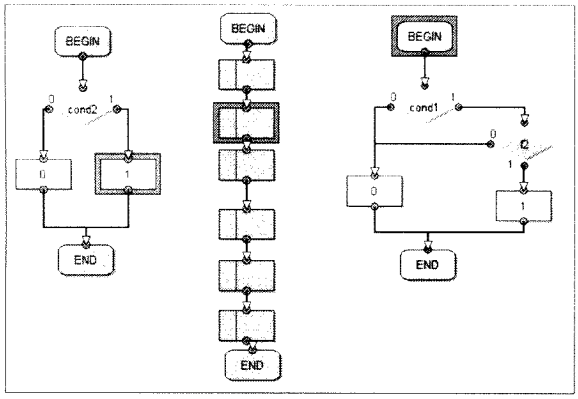


Figura 15. Espera de um sub-algoritmo pela terminação de outro para evitar a recursividade.

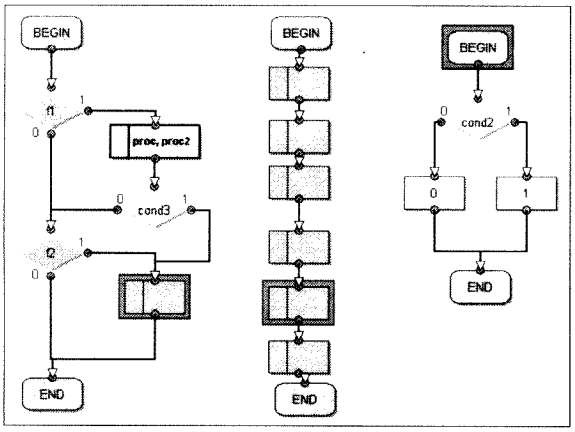


Figura 16. Invocação de f2 por f1 após a terminação da primeira.

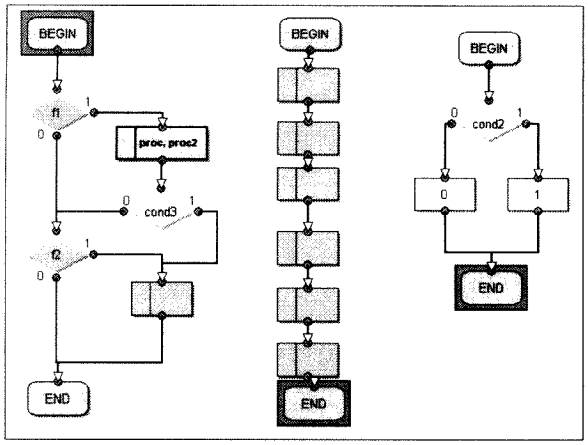


Figura 17. Terminação de proc e f2.

0	0	1	1	2	3
				3	5
				4	1

Figura 18. Estado do stack bidimensional relativamente ao estado da simulação da fig. 17.

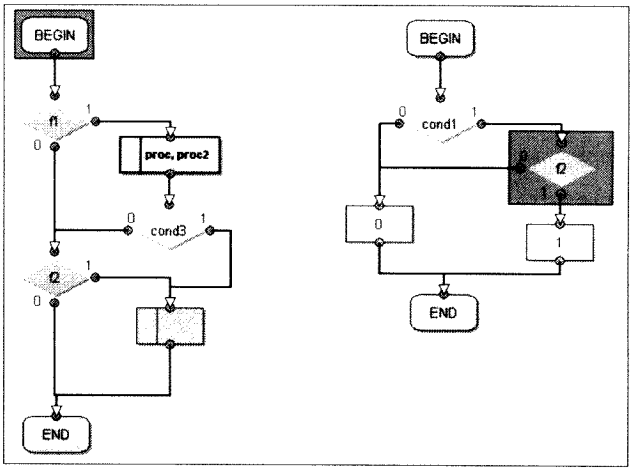


Figura 19. Retorno de f2 ao sub-algoritmo invocador f1.

Na situação apresentada na fig. 19 temos à esquerda o algoritmo principal e à direita temos f1. De notar que o primeiro permanece com o estado Begin activo porque o nodo que a este está ligado especifica a invocação da função f1 e que neste momento está a ser executada. Desta forma, evitando a reentrância, este sub-algoritmo terá de esperar que o da direita termine. Neste estado da simulação podemos verificar pela fig. 19 que a largura do *stack* bidimensional diminuiu, pois temos agora apenas dois sub-algoritmos a executar e um deles no segundo nível de hierarquia.

0	0	2	3
		3	5

Figura 20. Estado do stack bidimensional relativamente ao estado da simulação da fig. 19.

No final desta simulação podemos verificar que as dimensões máximas atingidas pela memória de *stack* foram 3x3, o que significa que obtivemos no máximo 3 invocações hierárquicas e 3 sub-algoritmos a executar em paralelo. Caso fosse permitida a reentrância as dimensões máximas obtidas certamente teriam sido superiores. De qualquer forma, a ferramenta GraphBuilder permite-nos ter conhecimento acerca do tamanho real do *stack* que devemos utilizar, caso estejamos na presença de um algoritmo de controlo simples, ou efectuar uma estimativa das dimensões necessárias ao correcto funcionamento do algoritmo na unidade de controlo pretendida.

J. A interacção com o utilizador

Durante a simulação o utilizador dispõe de uma barra de ferramentas, que no caso da fig. 9 pode ver-se em estado flutuante sobre o fundo da janela principal, onde se encontram todas as facilidades permitidas neste processo. É aqui que se inicia e termina a simulação e se simula a passagem de um ou mais ciclos de relógio, no caso da simulação passo a passo ou automática, respectivamente. No caso da simulação automática existe também a possibilidade de introdução de pontos de paragem (*breakpoints*) sobre os nodos de cada sub-algoritmo ou a pausa/interrupção do processo mediante a actuação de um dos botões da barra de ferramentas.

Todos os sinais de entrada e de saída da unidade de controlo considerada podem ser visualizados em janelas flutuantes, assim como o *log* da simulação. Inicialmente todos os sinais de entrada são colocados a zero mas durante a simulação o utilizador pode ir actuando sobre estes sinais binários entre cada ciclo de relógio, ou seja, antes de cada passo da simulação, para verificar o comportamento do algoritmo de controlo mediante as combinações binárias de entrada que desejar.

IV. UMA APLICAÇÃO PRÁTICA

O objectivo dos HiParaGraphs é disponibilizar um método fácil, rápido e eficaz de especificar unidades de controlo digitais. O CAN – *Controller Area Network*, é um protocolo de comunicação série que suporta a implementação de aplicações distribuídas e de tempo real e que é bastante utilizado nos sistemas electrónicos de automóveis. Estamos a falar de um barramento, num sistema *multi-master*, onde todos os nodos estão pendurados e são capazes de enviar e receber informação do barramento. Uma das grandes vantagens do protocolo CAN é a sua grande fiabilidade na transmissão. A quantidade de erros enviada por um nodo vai sendo medida e registada. Se o número de erros produzidos for grande o nodo será retirado do barramento. Este procedimento é possível devido à inexistência de endereços nos nodos e por isso estes podem ser acrescentados e retirados do barramento sem que haja interferências no sistema. Os endereços são substituídos por um sistema de envio de mensagens de diferentes prioridades. Cada mensagem CAN pode transmitir 0 a 8 bytes de informação (dados) mas este comprimento pode ser estendido utilizando segmentação, sendo a taxa máxima de transmissão especificada a 1Mbit/s.

Vários nodos podem requisitar simultaneamente o barramento para o envio de mensagens. Um transmissor envia uma mensagem a todos os nodos CAN (*broadcasting*) e cada nodo decide, baseando-se no identificador recebido se deve ou não processar aquela mensagem. Este identificador já determina a prioridade da mensagem no acesso ao barramento.

Os vários tipos de mensagens são:

- DATA FRAME – de dados, enviada por um transmissor a todos os nodos receptores.
- REMOTE FRAME – transmitida por um nodo para requisitar a transmissão de uma mensagem de dados com o mesmo identificador.
- ERROR FRAME – é transmitida por um nodo quando este detecta um erro no barramento.
- OVERLOAD FRAME – é utilizada para introduzir um atraso entre as mensagens DATA FRAME e REMOTE FRAME.

As mensagens DATA e REMOTE são separadas das restantes por um INTERFRAME SPACE que consiste numa mensagem de pelo menos três bits consecutivos a 1.

Uma mensagem do tipo DATA é composta por sete campos diferentes:

1. *Start of Frame* – composto por um bit igual 0 que delimita o início da mensagem.
2. *Arbitration* – campo de arbitragem, composto por um identificador de 11 bits e um bit RTR.
3. *Control* – composto por 6 bits, dois reservados e quatro para indicar o comprimento dos dados.
4. *Data* – campo que contém todos os bits de dados, desde 0 a 8 bytes.
5. *CRC* – contém uma sequência CRC (Cyclic Redundancy Code) e um bit delimitador.
6. *ACK* – campo de dois bits.
7. *End of Frame* – campo de 7 bits iguais a 1.

Uma mensagem do tipo REMOTE é equivalente à anterior pois possui todos estes campos à excepção do *Data*.

O protocolo CAN possui uma unidade de controlo para transferência de mensagens que os HiParaGraphs são capazes de descrever. Esta unidade possui características hierárquicas e paralelas e utiliza as condições de saída dos HiParaGraphs para especificar o comportamento do sistema em situações de erro. A descrição comportamental da unidade de controlo foi dividida em vários módulos. Existem três sub-algoritmos que são executados em paralelo desde o início e que estão representados nas figuras 21, 22 e 23. O primeiro dedica-se ao controlo da transferência das mensagens no barramento, o segundo ao controlo sobre a ocorrência de erros e o terceiro ao controlo do papel de emissor ou receptor de cada nodo CAN existente no barramento. Os dois últimos correspondem a duas máquinas de estado simples enquanto o primeiro é composto por quatro sub-algoritmos e possui hierarquia devido aos diferentes tipos de campos existentes numa mensagem, como foi apresentado acima.

Na fig. 21 estão visíveis todos os campos de uma mensagem, quer do tipo DATA quer REMOTE. O controlo da recepção/transmissão de cada bit daqueles campos constituídos por mais do que um bit é encapsulado noutro sub-algoritmo de controlo – temos portanto hierarquia. Entramos no campo de arbitragem quando é invocado hierarquicamente o procedimento DR_ARB (fig. 24a). Seguidamente passamos ao campo de controlo, invocando da mesma forma o procedimento DR_CTRL (fig. 24b). O

bit *rtr* dentro do nodo de teste que se segue determina se estamos perante uma mensagem do tipo DATA ou do tipo REMOTE. Os campos CRC e ACK estão representados pelos procedimentos DR_CRC (fig. 25a) e DR_ACK (fig. 25b) e tal como os anteriores também são invocados hierarquicamente. Todos os procedimentos são invocados com o indicador W, pois temos de esperar a sua terminação para passar ao campo seguinte da mensagem. Na parte inferior do sub-algoritmo está especificada a condição de saída que indica que se o sinal *error* estiver activo devemos sair deste algoritmo e invocar o FRM_ERROR (ver fig. 26a).

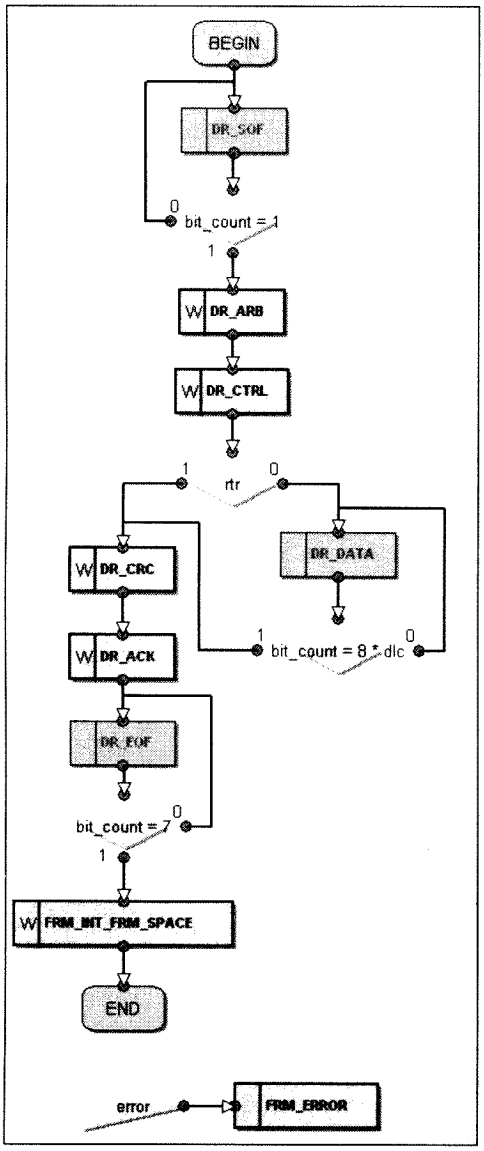


Figura 21. Sub-algoritmo pertencente à especificação da unidade de controlo do CAN responsável pelo controlo das transferências de mensagens.

Em qualquer instante um nodo CAN pode estar num dos três estados representados na fig. 22:

1. FLT_ERROR_ACTIVE – significa que o nodo pode enviar para o barramento mensagens de erro do tipo ACTIVE_ERROR_FLAG. Tal como podemos ver na figura pelo sinal de saída *error_flag* = 0.
2. FLT_ERROR_PASSIVE – significa que o nodo pode enviar para o barramento mensagens do tipo PASSIVE_ERROR_FLAG, pela actualização do sinal *error_flag* = 1.
3. FLT_BUS_OFF – neste estado o nodo não pode enviar mensagens para o barramento.

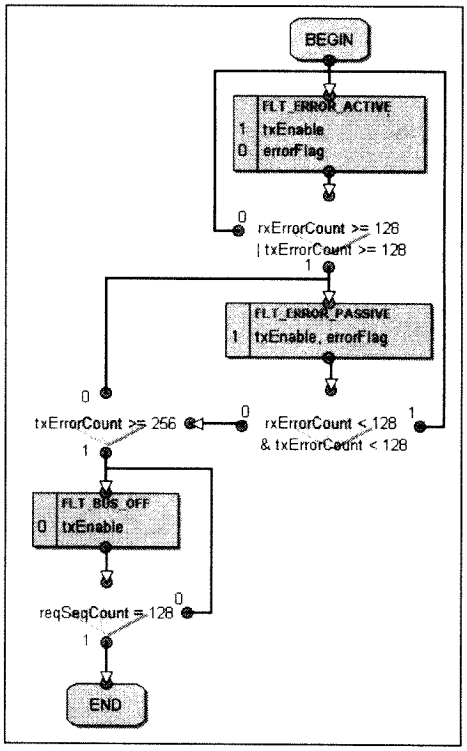


Figura 22. Sub-algoritmo pertencente à especificação da unidade de controlo do CAN responsável pelo controlo sobre a ocorrência de erros no barramento.

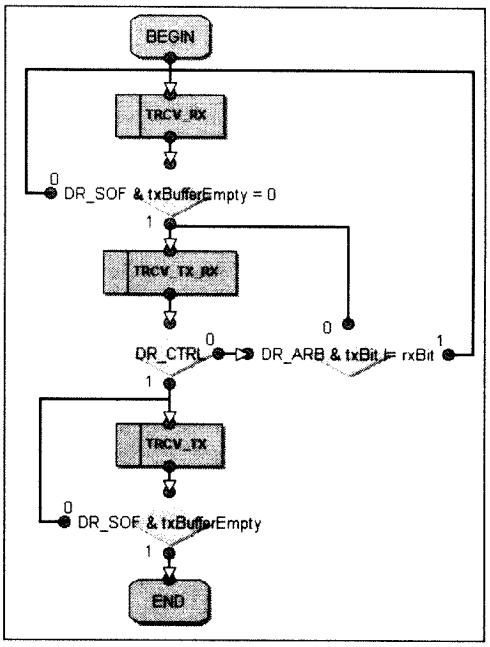


Figura 23. Sub-algoritmo pertencente à especificação da unidade de controlo do CAN responsável pelo controlo sobre a emissão e recepção de mensagens de cada nodo.

Na fig. 23 visualizamos também três estados denominados:

1. TRCV_RX – que significa que o nodo CAN está apenas em modo de recepção,
2. TRCV_TX_RX – significa que o nodo é capaz de transmitir e receber mensagens do barramento,
3. TRCV_TX – significa que o nodo está apenas em modo de transmissão.

O sub-algoritmo representado na fig. 23 explica como um nodo pode transitar entre estes três estados, pela verificação das condições especificadas dentro dos nodos de teste. Os nodos de teste com as condições escritas a letra maiúscula, como por exemplo DR_CTRL e DR_ARB pretendem testar se os respectivos sub-algoritmos com este nome estão activos.

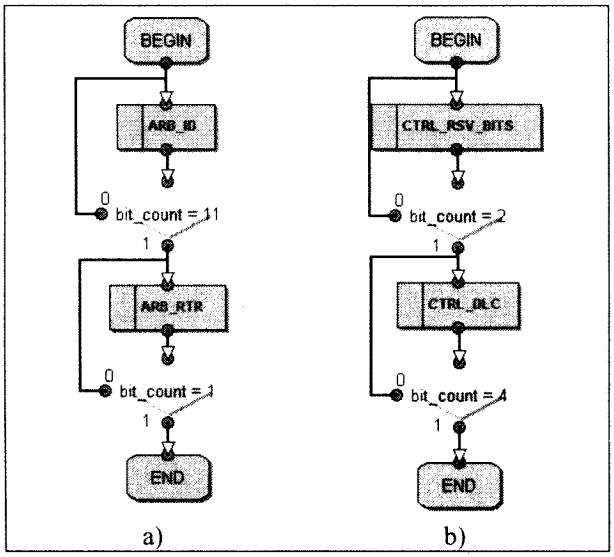


Figura 24. Sub-algoritmos pertencentes à especificação da unidade de controlo do CAN que especificam os procedimentos a) DR_ARB e b) DR_CRTL.

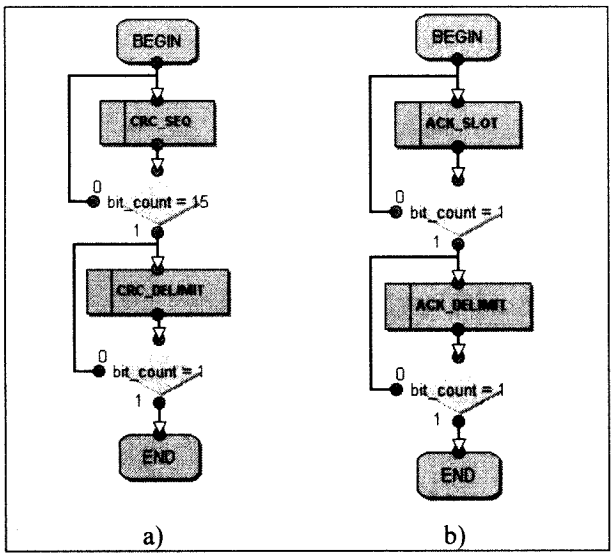


Figura 25. Sub-algoritmos pertencentes à especificação da unidade de controlo do CAN que especificam os procedimentos a) DR_CRC e b) DR_ACK.

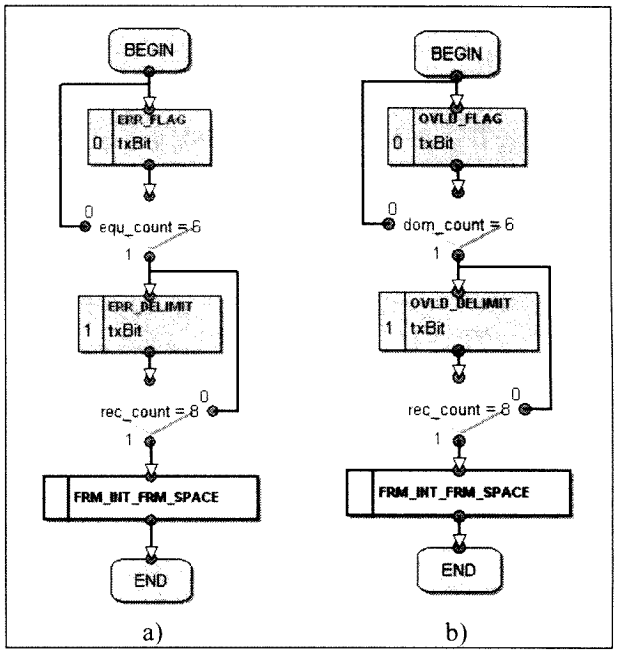


Figura 26. Sub-algoritmos pertencentes à especificação da unidade de controlo do CAN que especificam os procedimentos a) FRM_ERROR e b) FR_OVERLOAD.

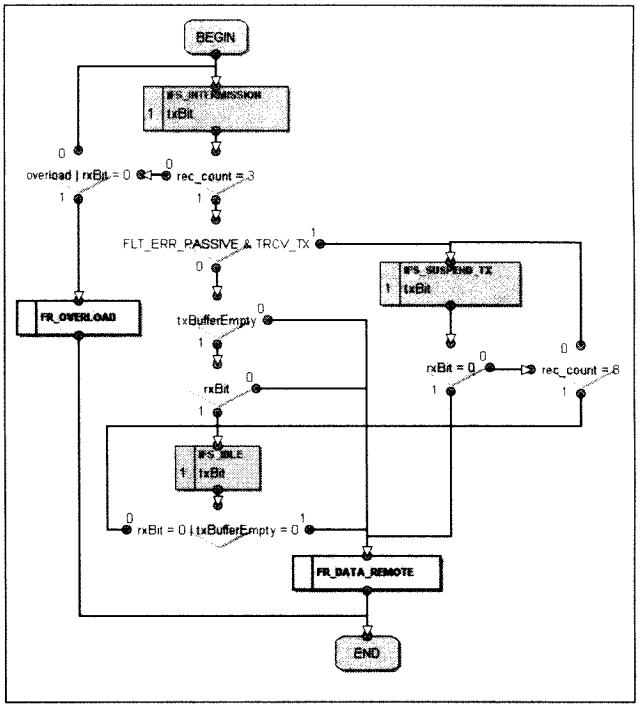


Figura 27. Sub-algoritmo pertencente à especificação da unidade de controlo do CAN que especifica o procedimento FRM_INT_FRM_SPACE.

Esta unidade de controlo foi utilizada como um estudo de caso de forma a avaliar as capacidades de especificação dos HiParaGraphs e eliminar as suas deficiências, acrescentando-lhes alguma funcionalidade.

A visualização do nome do estado associado a um nodo de activação foi uma das facilidades visuais (opcionais) introduzidas na aplicação pois revelou-se útil na compreensão da descrição comportamental dos módulos apresentados, como por exemplo os das figuras 22, 23, 24 e 25, pois como não activam nenhum sinal de saída ficariam vazios e, consequentemente, sem nenhum significado visível ao projectista. O nome do estado aparece sempre escrito no topo do nodo de activação, por cima de todos os sinais, com cor diferente e em maiúsculas. No caso da fig. 25a temos visualizados os estados CRC_SEQ e CRC_DELIMIT.

A introdução dos nodos com condições de saída nos HiParaGraphs deveu-se ao estudo deste exemplo de especificação. Como se pode ver na fig. 21, existe uma condição de saída na parte inferior do sub-algoritmo, que especifica o sinal *error*. Isto significa que se em qualquer estado do circuito esta condição se verificar, ou seja, se *error* estiver activo, o sub-algoritmo FRM_ERROR será imediatamente invocado. Esta forma de saída de um sub-algoritmo dependente de uma condição evita o teste desta em todos os estados possíveis do sub-algoritmo, o que complicaria demasiado o aspecto gráfico, a conversão para código VHDL e a respectiva implementação do circuito.

Os sinais de entrada que estão especificados dentro dos nodos condicionais mostram expressões, como por exemplo, *rtr*. A avaliação do valor lógico de um sinal binário dentro destes nodos é efectuada determinando se este tem o valor verdadeiro (1). No caso das expressões *bit_count = 8 * dlc*, *bit_count = 1* e *bit_count = 7* da fig. 21, a avaliação é efectuada de outra forma. Embora estas sejam mais explícitas para quem especifica a unidade de controlo, terão de ser substituídas por outros sinais binários aquando da conversão do algoritmo em código VHDL. Esta forma livre de especificar condições dentro dos nodos é útil e intuitiva ao projectista, pois demonstra uma interligação entre a unidade de controlo e a unidade de execução de um sistema, mas este terá de analisar todos os sinais de entrada (que é uma tarefa fácil utilizando o GraphBuilder) para posteriormente atribuir outros nomes antes da conversão para VHDL, pois poderá haver incompatibilidade entre os nomes das condições do tipo *bit_count = 8 * dlc*, e a sintaxe do VHDL.

A existência evidente de uma interligação entre a unidade de controlo e a unidade de execução é a especificação de sinais de entrada do tipo *bit_count = x*. Assumem-se aqui que existem vários contadores de bits, que vão sendo recebidos do barramento e contados. Quando for atingido um determinado valor que é especificado dentro do nodo de teste então a condição verifica-se. Devem existir quatro contadores diferentes de bits recebidos do barramento, que contam:

1. *bit_count* – todos os bits independentemente do seu valor lógico;
2. *rec_count* – todos os bits com o valor lógico 1;
3. *dom_count* – todos os bits com o valor lógico 0;
4. *equ_count* – todos os bits consecutivos iguais.

V. CONCLUSÕES

A especificação modular de algoritmos de controlo através de HiParaGraphs suporta explicitamente a hierarquia e o paralelismo. Com este método pretende-se automatizar o projecto de unidades de controlo partindo de uma especificação facilmente extensível e sintetizável. Os HiParaGraphs são uma descrição gráfica formal independente do modelo de implementação do circuito. A ferramenta de software GraphBuilder implementa todo o suporte para esta linguagem, desde a construção gráfica até à simulação e conversão em código VHDL.

A simulação dos algoritmos de controlo especificados por HiParaGraphs prevê a existência de hierarquia, paralelismo e ambos em simultâneo, recursividade e reentrância, podendo a última ser utilizada como opção devido a possíveis limitações de implementação por parte do projectista. A simulação, tal como foi apresentada neste artigo pode ser utilizada para efeitos de depuração do circuito de forma a serem detectados erros no comportamento deste antes de ser convertido em VHDL e posteriormente sintetizado, facilitando assim, mas não evitando, a depuração e teste do circuito após a sua implementação.

Como estudo de caso para evolução desta linguagem, assim como da ferramenta GraphBuilder, foi utilizada a especificação da unidade de controlo do protocolo CAN. Este estudo resultou na introdução de um certo grau de liberdade na especificação:

- do teste de sinais de entrada, que pode ser unicamente um sinal binário ou mais do que um afectados por uma função Booleana (*and, or,* ou *xor*). Por outro lado, se as condições especificadas tiverem o mesmo nome de um sub-algoritmo isto significa que estamos a testar se este sub-algoritmo se encontra activo naquele instante.
- na visualização dos estados no topo dos nodos de activação e
- na introdução de um novo tipo de nodo com condições de saída de sub-algoritmos.

Assim, podemos concluir que os HiParaGraphs são uma linguagem gráfica cada vez mais poderosa e com possibilidades de evolução na construção de unidades de controlo digitais. O ambiente GraphBuilder é uma ferramenta de trabalho que tem os HiParaGraphs como suporte, disponibilizando-lhes todas as funcionalidades e interface até agora apresentadas. A unidade de controlo deste protocolo pode ser construída e simulada no ambiente GraphBuilder e facilmente convertida em código VHDL comportamental sintetizável.

REFERÊNCIAS

[1] Andreia Melo, Valery Sklyarov, António Ferrari, *HiParaGraphs, uma Linguagem de Especificação de Algoritmos de Controlo Paralelos e Hierárquicos*, Electrónica e Telecomunicações, Vol. 3, Nº 3, pp. 214-221, Janeiro de 2001.

[2] Andreia Melo, Valery Sklyarov, António Ferrari, *Specification and Implementation of Hierarchical and Parallel Control Algorithms Using HiParaGraphs*, aceite para publicação nas actas de CAD DD'2001 - 4th International Conference on "Computer-Aided Design of Discrete Devices", Minsk, Bielorrúsia, Novembro 2001.

[3] Andreia Melo, *Especificação, Optimização e Teste de Algoritmos de Controlo Hierárquicos*, Dissertação de Mestrado em Engenharia Electrónica e de Telecomunicações, Universidade de Aveiro, Janeiro 2000.

[4] A. Melo, V. Sklyarov, *Ambiente Integrado para Especificação, Projecto e Verificação de Unidades de Controlo em FPGAs*, Electrónica e Telecomunicações, vol. 2, n.º 4, Janeiro 1999, pp. 477-485.

[5] Road vehicles – *Interchange of digital information – Part 1: Controller area network data link layer and medium access control*; ISO 11898.

[6] Road vehicles – *Controller area network (CAN) – Part 2: High-speed medium access unit*; ISO 11898.

[7] David Harel, *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming, Nº8, pp. 231-271, 1987.

[8] Tadao Murata, *Petri Nets: Properties, Analysis and Applications*, Proceedings of the IEEE, Vol. 77, Nº4, pp.541-590, Abril 1989.

[9] A. D. Zakrevskij, *Parallel Algorithms for Logical Control*, Institute of Engineering Cybernetics of NAS of Belarus, Minsk, 1999.

[10] Diestel R., *Graph Theory*, Springer, New York, 2ª ed., 2000.

Programação e Gestão de Interfaces em Visual C++

Andreia Melo

Resumo – **O interface gráfico de uma aplicação de software tem hoje em dia cada vez mais importância. Uma aplicação é tanto mais atractiva quanto melhor for a sua interface, eficiência e facilidade de utilização. Neste artigo pretende-se fornecer os princípios básicos e os primeiros passos para construir uma aplicação de software numa linguagem orientada por objectos como é o Visual C++.**

Abstract – **The graphical interface of a software application is a factor of increasing importance nowadays. An application is much more attractive and accepted as good as it is its interface, efficiency and easy to use. This paper is used to provide the basic principles and the first steps for building a software application using an object oriented language such as Visual C++.**

I. INTRODUÇÃO

Nos dias de hoje, qualquer tipo de aplicação executável em computador possui um interface gráfico. A utilização do computador é um processo que já se tornou banal e essencial e que faz parte da vida quotidiana de pessoas de todas as faixas etárias que utilizam programas para os mais diversos fins. A nível profissional, onde acedem a bases de dados em computadores terminais e efectuam operações sobre elas, em actividades associadas ao lazer, como por exemplo para aceder aos mais variados tipos de informação disponibilizada na Internet e jogos de computador. Todos estes gestos assumem uma interacção entre a pessoa e o computador e devem parecer o mais intuitivos possível, ou seja, qualquer tipo de pessoa, independentemente das suas características, ao utilizar o computador para um determinado fim não deve dispender o seu tempo a tentar compreender como pode interagir com uma dada aplicação, como por exemplo à procura dos comandos de que necessita, mas concentrar-se apenas na operação que deseja efectuar. Neste artigo vamos considerar aquele grupo de pessoas que pretendem construir as aplicações ou ferramentas de trabalho e a forma como o fazem, isto é, como devem utilizar os recursos de interface já existentes para criar uma aplicação executável, fácil de utilizar, eficiente e por isso atractiva. A linguagem de programação que aqui será abordada é o Visual C++, embora a linguagem Visual Basic [1] seja a mais popular para este efeito.

Este artigo divide-se em sete secções A primeira consiste nesta introdução. A segunda secção mostra quais os primeiros passos na construção de uma aplicação utilizando o Visual C++. Esta linguagem dispõe de uma filosofia de construção de um programa, a que foi chamada arquitectura documento/vista que será detalhada na terceira secção. A quarta secção descreve os recursos de interface com o utilizador e forma como podem ser utilizados. Na quinta secção é mostrado como se podem criar objectos personalizados, visualizá-los no ecrã e interagir com eles. Os diferentes tipos de aplicações onde estes princípios de interface podem ser utilizados são apresentados na sexta secção. A secção sete finaliza este artigo com algumas conclusões.

II. CONSTRUÇÃO DE UMA APLICAÇÃO

A primeira operação a efectuar quando pretendemos construir uma aplicação em Visual C++ no ambiente de desenvolvimento Microsoft Visual Studio 6.0 é criar um novo projecto. Neste passo surgem-nos várias opções mas a que aqui será seguida é a aplicação executável que utiliza a biblioteca MFC (*Microsoft Foundation Classes*), chamada **MFC AppWizard(exe)** e é um processo guiado através de um assistente. A biblioteca de classes que é aqui utilizada dispõe de todas as funções necessárias à construção de um interface. Para uma melhor familiarização com esta biblioteca sugere-se a consulta de [2] ou versões mais actualizadas.

A. *Tipos básicos de interfaces de aplicações*

Existem três tipos de interfaces de aplicações que podem ser criadas: SDI – *Single Document Interface*, MDI – *Multiple Document Interface* e *Dialog based* (baseada apenas numa caixa de diálogo). A primeira utiliza a janela principal da aplicação para mostrar o documento e só apenas um documento pode ser visualizado de cada vez. No próximo passo do assistente é dado a escolher se pretendemos suporte para bases de dados e se tal for desejado, deveremos indicar a fonte dos dados que irão ser utilizados.

Ao nível dos recursos de interface com o utilizador são disponibilizadas pelo assistente o menu, a barra de ferramentas, a barra de estado, o suporte para impressão e visualização prévia do documento a imprimir, o suporte de ajuda (*Context-sensitive Help*) e os controlos 3D. O aspecto das barras de ferramentas, *Normal* e *Internet Explorer ReBars* são também opções disponíveis. Aconselha-se a utilização da segunda pois actualmente todas as aplicações de trabalho utilizam este modelo de apresentação, isto é, parecem não existir os botões, que

são realçados apenas quando o utilizador coloca o apontador do rato sobre o objecto desenhado sobre a barra. Este tipo de barra de ferramentas permite além disto a colocação de outros tipos de controlos, tais como listas (*combo boxes*), *menus popup*, etc.

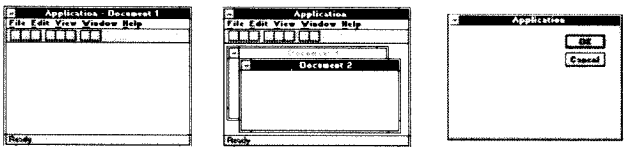


Figura 1. Diferentes tipos de interfaces básicas de uma aplicação: (a) SDI, (b)MDI e (c) caixa de diálogo.

B. As classes principais e a sua função

Terminando a função do assistente para a criação da aplicação executável, assumindo que foi escolhido o tipo de interface SDI ou MDI, verifica-se que foram construídas automaticamente cinco classes. *CAboutDlg*, que é utilizada para invocar e visualizar uma caixa de diálogo com informação específica acerca da aplicação, como por exemplo o seu nome, versão e autor(es). A classe *CExemploApp* é derivada da classe *CWinApp* que é utilizada como classe base sempre que se cria uma aplicação em Windows. Esta fornece as funções para inicialização do objecto que corresponde à própria aplicação, e cada instância desta, e para executá-la. Sempre que se utilize a biblioteca MFC só pode existir um objecto derivado da classe *CWinApp*. O objecto derivado desta classe, isto é, do tipo *CExemploApp*, deve ser declarado globalmente pois é construído ao mesmo tempo que outros objectos globais e já deverá existir quando o Windows invoca a função *WinMain*, fornecida pela MFC. Deve-se também redefinir a função *InitInstance* da classe *CWinApp* dentro da nossa classe *CExemploApp*. É aqui que se define a utilização dos controlos 3D (pela invocação da função *Enable3dControls*), que se define o template do documento que irá ser utilizado nesta aplicação e onde são invocadas as funções *ShowWindow* e *UpdateWindow* para visualização da janela principal.

III. CRIAÇÃO DE TEMPLATES DE DOCUMENTOS

Quando uma aplicação é executada em Windows o utilizador interage com documentos apresentados em *frames*. A *frame* de uma janela com um documento possui basicamente duas componentes: a *frame* propriamente dita, isto é, a delimitação da janela e o seu conteúdo. São estas *frames* que podem ser do tipo SDI ou MDI. No primeiro caso existe apenas uma enquanto no segundo existe um conjunto de janelas filhas que apresentam o conteúdo de diferentes documentos.

Os templates de documentos são utilizados para, durante a criação de um novo documento, através de um comando *New* ou *Open* do menu *File*, criar uma nova *frame* para visualizar esse documento. O construtor do template especifica o tipo do documento, a janela e a vista pois estes são os argumentos a ele passados. No caso de uma aplicação de interface SDI o código apresentado na fig. 2 ilustra a criação do template, invocando a função *AddDocTemplate* que neste caso é invocada apenas uma vez pois existe só um tipo de documento.

```
AddDocTemplate(new CSingleDocTemplate(
                IDR_MAINFRAME,
                RUNTIME_CLASS(CExemploDoc),
                RUNTIME_CLASS(CMainFrame),
                RUNTIME_CLASS(CExemploView));
```

Figura 2. Criação do template do documento para uma aplicação de interface SDI.

Os argumentos passados ao construtor consistem num identificador de recursos (*resource ID*) associado ao tipo de menu e aceleradores utilizados na aplicação, e a três invocações da macro RUNTIME_CLASS que devolvem um objecto *CRuntimeClass* do tipo especificado no seu argumento. Os três objectos *CRuntimeClass* passados ao construtor do template do documento fornecem a informação necessária à criação de novos objectos de classes que são especificadas durante o processo de criação do documento.

Na fig. 3 é apresentado o código de criação de um template de documento no caso das aplicações MDI. Neste tipo de interface a função *AddDocTemplate* tem de ser invocada para cada tipo de documento criado, como por exemplo, documentos de texto, de gráficos, etc.

```
AddDocTemplate ( new CMultiDocTemplate(
            IDR_SCRIBTYPE,
            RUNTIME_CLASS(CExemploDoc),
            RUNTIME_CLASS(CChildFrame),
            RUNTIME_CLASS(CExemploView)));
```

Figura 3. Criação do template do documento para uma aplicação de interface MDI.

O sistema operativo Windows é responsável pela maior parte da interacção do utilizador com a *frame*, incluindo as funções de deslocamento, redimensionamento, fecho, minimização e maximização. O trabalho do projectista reduz-se à gestão do conteúdo da *frame*.

IV. ARQUITECTURA DOCUMENTO/VISTA

Por defeito as aplicações que se constroem baseadas na biblioteca MFC utilizam um modelo de programação que separa os dados da maneira como estes são apresentados ao utilizador, separando também a forma como este interage com eles. Neste modelo, o objecto que corresponde ao **documento** MFC lê e escreve dados de forma persistente. No entanto, existe um novo objecto a que foi chamada **vista**, que manipula a visualização dos dados, como por exemplo o desenho dos dados numa janela para que o utilizador os possa seleccionar e editar. Além desta função a vista também é responsável por comunicar ao documento quaisquer alterações a este

efectuadas. As classes mais importantes que servem de suporte a este modelo são a:

- *CDocument*, cujos objectos são usados para guardar e controlar todos os dados do programa;
- *CView*, cujos objectos servem para apresentar os dados do documento e gerir a interacção entre os dados e o utilizador;
- *CFrameWnd*, cujo objecto fornece a *frame* existente numa ou mais vistas de um documento;
- *CDocTemplate* (ou *CSingleDocTemplate* ou *CMultiDocTemplate*), cujos objectos coordenam a existência de um ou mais tipos de documentos e gerem a criação correcta do documento, vista e *frame* associadas ao tipo de documento indicado.

Este modelo de programação, que se baseia no conceito de separação do documento da vista, revela-se bastante eficiente nas aplicações em que se pretendem várias vistas do mesmo documento, como por exemplo as folhas de cálculo ou a visualização de gráficos. Suponhamos que o utilizador da aplicação pretende visualizar os dados ou na forma de uma folha de cálculo ou na forma de um gráfico. E suponhamos agora que os pretende ver simultaneamente de forma a comparar os resultados. A forma mais fácil de implementar esta funcionalidade é dividir estes dois tipos de vistas em diferentes *frames*, ou dentro da mesma janela, dividida por um separador (*splitter*), cuja função será explicada na secção seguinte. Por outro lado, o utilizador pode desejar alterar os seus dados e ver essas alterações imediatamente reflectidas no gráfico. Podemos assim concluir que o modelo de programação que estamos aqui a considerar é o ideal para implementar este tipo de operações. Assim, o código comum a todas as vistas, como os métodos de cálculo sobre os dados, fica contido no documento que, por sua vez, necessita de actualizar todas as vistas sempre que ocorram alterações nos dados. Utilizando a biblioteca MFC as vistas criadas devem ser classes derivadas da *CView* e, neste exemplo que estamos a considerar, ambas devem estar associadas ao mesmo documento (que contém os dados ou obtem-nos a partir de uma base de dados). Na fig. 4 estão apresentadas as várias formas de associar diferentes vistas ao mesmo documento.

Para resumir seguem-se alguns tópicos relacionados com a função da classe que implementa o documento e a vista.

As tarefas principais de um documento MFC são:

- O armazenamento dos dados,
- A inicialização dos documentos e das vistas,
- A inicialização daquilo que adicionamos às classes do documento e das várias vistas,
- Limpar os documentos e as vistas e
- Criar múltiplos tipos de documentos.

A vista MFC tem como funções:

- Mostrar e/ou desenhar os dados,
- Interpretar a interacção do utilizador,
- Efectuar as operações de *scrolling* e dimensionamento,
- Criar múltiplas vistas sobre um documento,
- Utilizar uma vista com diferentes tipos de controlos, tal como numa caixa de diálogo e
- Utilizar uma vista para mostrar os vários campos de informação de uma base de dados.

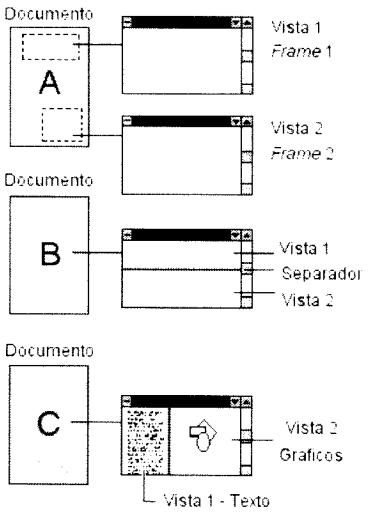


Figura 4. Diferentes vistas sobre o mesmo documento.

V. RECURSOS DE INTERFACE COM O UTILIZADOR

Existem vários tipos de interfaces gráficos, como por exemplo as janelas, barras de ferramentas, menus, aceleradores, separadores e controlos, a que chamamos recursos de interface.

C. *Tipos de janelas*

As janelas de uma aplicação podem ser dos mais variados tipos e cada uma delas possui uma *frame*, ou seja, uma delimitação cuja função pode variar dependendo da forma de derivação da classe *CFrameWnd*. Ao falar em janelas no sistema operativo Windows estamos a referirmo-nos a tudo que conseguimos visualizar, pois afinal tudo são janelas, desde a janela principal da aplicação a uma caixa de diálogo e até um simples controlo. Tudo isto depende apenas do programador quando lhe atribui uma forma, interface e funcionalidade.

D. *Separadores*

Quando falamos da janela que apresenta o conteúdo dos documentos que são utilizados, podemos incluir um separador para as diferentes vistas, como se pode ver na fig. 4 relativamente ao documento B (separador horizontal). A função do separador é dividir as janelas em vários painéis, cada um com a possibilidade de mostrar diferentes partes do documento e de diferentes maneiras. Os separadores, horizontais ou verticais, podem ser criados **estatica** ou **dinamicamente**.

Se for de forma estática a janela possui, definido à partida no programa, um número fixo de painéis, normalmente associados a classes diferentes. É necessário

um objecto do tipo *CSplitterWnd* declarado normalmente na classe que derivamos da *CMDIChildWnd*. Para criar um separador de forma estática é necessário redefinir a função *OnCreateClient* da respectiva classe e aqui invocar a função *CreateStatic* a partir do objecto a que corresponde o separador. Nesta função são passados como argumentos o número de linhas e de colunas em que queremos dividir a nossa janela de visualização. O próximo passo é criar todos os painéis invocando a função *CreateView* do separador. É nesta função que especificamos o painel a que nos referimos através do índice da linha e da coluna a que lhe corresponde, o seu tamanho mínimo e a classe, normalmente derivada da *CView*, a que vai ser associado. A escolha da vista inicialmente activa é efectuada invocando a função *SetActiveView*, à qual se fornece o painel da vista que pretendemos. Na fig. 5 é apresentada uma forma de divisão da janela de aplicação, em duas linhas e uma coluna. No entanto, a linha inferior é também dividida em duas vistas diferentes.

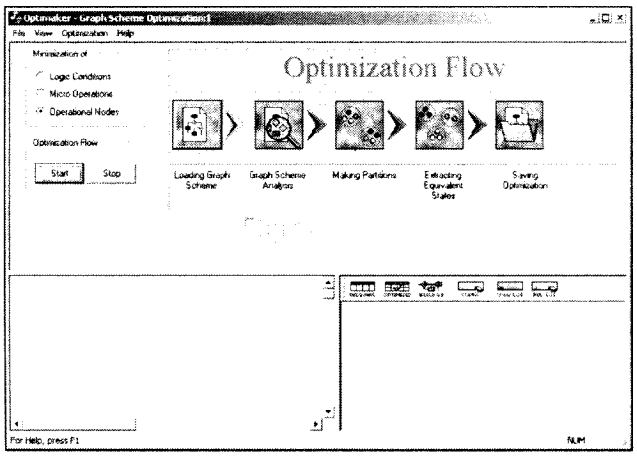


Figura 5. Exemplo de aplicação com três tipos de vistas diferentes separadas por 2 *splitters*.

Seguidamente é apresentado o código necessário para conseguir este tipo de interface. Existem dois separadores, instâncias da classe *CSplitterWnd*, que se chamam `m_split1` e `m_split2`. O primeiro divide a janela horizontalmente, pois podemos ver que são passados os argumentos 2 e 1 à função *CreateStatic* que correspondem ao número de linhas e de colunas, respectivamente. Tal como já foi explicado acima, deve-se invocar a função *CreateView* a partir da instância `m_split1` para criar a vista que podemos ver na parte superior da janela da fig. 5. A classe associada a esta vista foi chamada *CControlView* apenas porque contém vários tipos de controlos. No entanto, não podemos fazer este mesmo procedimento para as vistas inferiores, ou seja, não podemos invocar a função *CreateView* novamente utilizando a instância `m_split1`. Em vez disso passamos agora a usar a instância `m_split2` e invocamos a função *CreateStatic* mas com uma diferença. No primeiro caso, o ponteiro *this* foi passado como primeiro argumento à função *CreateStatic* porque este argumento corresponde a um ponteiro para a janela pai do separador, que no caso do `m_split1` é a *frame* da janela principal e daí a utilização do *this*. No caso do `m_split2` a janela pai é o próprio `m_split1`, que também é uma janela. Assim, na invocação da função *CreateStatic* é necessário indicar o ponteiro para o `m_split1` (`&m_split1`), o número de linhas desta nova janela (`1`) e o número de colunas (`2`). Para indicar que estamos a dividir a segunda linha de `m_split1` temos de indexá-la (de notar que os índices começam sempre por 0) utilizando a função *IdFromRowCol*, onde 1 corresponde ao índice da linha e 0 ao índice da coluna. A criação das vistas para `m_split2` é feita pelo método normal, já explicado anteriormente. Neste extracto de código terminamos por activar a vista relativa à parte superior da janela pela invocação da função *SetActiveView*.

```
m_split1.CreateStatic(this,2,1)

m_split1.CreateView(0, 0,
  RUNTIME_CLASS(CControlView),
  CSize(0,270), pContext))

m_split2.CreateStatic(
     &m_split1, 1, 2,
     WS_CHILD|WS_VISIBLE,
     m_split1.IdFromRowCol(1, 0)))

m_split2.CreateView(0, 0,
     pContext -> m_pNewViewClass,
        CSize(600, 0), pContext)

m_split2.CreateView(0, 1,
     RUNTIME_CLASS(CFrameTable),
        CSize(0, 100), pContext)

SetActiveView(
        (CView*)m_split1.GetPane(0, 0));
```

Figura 6. Extracto de código necessário à criação das três vistas da fig. 5 utilizando separadores estáticos.

Os separadores criados dinamicamente só podem ser utilizados sobre a mesma vista (a classe da vista tem de ser a mesma). Isto serve apenas para visualizar diferentes partes do documento mas com a mesma interface (tal como se passa na aplicação Microsoft Word).

O separador é um recurso de interface bastante útil porque sendo móvel, pode ser arrastado por intermédio do rato, permitindo ao utilizador aumentar a área do ecrã disponível para visualizar a sua informação. Por outro lado, se estivermos a falar de utilizadores inexperientes, o separador é um recurso que inicialmente pode esconder, em certos casos, os controlos colocados numa das vistas da aplicação.

E. Menus e Aceleradores

Os menus de uma aplicação desempenham um papel importantíssimo pois é aqui que todas as tarefas efectuadas pela aplicação têm de ser catalogadas e separadas de forma a que o utilizador, por mais

inexperiente que seja, consiga encontrar facilmente todos os comandos que deseja. O menu principal de uma aplicação situa-se geralmente no topo da janela e deve estar organizado da forma mais *standard* possível. Por exemplo, quando pretendemos manipular ficheiros, é usual utilizar um menu do tipo apresentado na fig. 7.

Figura 7. Ordenação *standard* das operações num menu.

Aconselha-se este método *standard* de ordenar a apresentação da funcionalidade pois assim estamos a diminuir o risco de confusão de um utilizador quando este passa de aplicação para aplicação. Inconsientemente a pessoa habitua-se a encontrar os menus em determinados sítios e por isso o programador deve ter isto em conta quando pretende uma boa aceitação da aplicação que projectou.

A cada opção disponibilizada dentro do menu deverão ser atribuídas teclas de atalho para que utilizadores mais experientes de uma dada aplicação as possam usar depois de uma familiarização com o programa, aumentando assim a rapidez de utilização e, consequentemente, a sua produtividade. A estes atalhos chamam-se aceleradores pois destinam-se precisamente a este efeito.

Com o aparecimento do sistema operativo Windows 2000, podemos ver que a maioria das aplicações Microsoft dispõem de menus que nos primeiros instantes em que são activados se visualizam apenas as opções mais frequentemente utilizadas. Esta é uma evolução da interface que pretende eliminar do ecrã informação à partida desnecessária ao utilizador frequente mas que poderá causar alguma confusão, independentemente da experiência da pessoa, pois os comandos mudam de sítio e a maioria fica escondida, dificultando a procura de um comando que já há muito não é utilizado. Para substituir este método e facilitar realmente a procura dos comandos utilizados mais frequentemente deverão ser disponibilizadas as barras de ferramentas que serão descritas na secção seguinte.

Contudo, existe outro tipo de menus, denominados *popup*, que são activados com o botão direito do rato. Estes são normalmente personalizados dependendo do tipo de objecto apontado pelo rato. Seguidamente é apresentado um extracto de código escrito em Visual C++ (fig. 8) que ilustra a criação de um menu *popup*. Para que este seja visualizado ao apontar certo objecto, cuja classe seja derivada da *CObject*, o código apresentado deve ser escrito dentro da função *OnRButtonDown* que responde ao evento do clique do botão direito do rato. Esta função deverá ser redefinida na classe da vista onde o objecto se encontra. Vamos assumir que temos uma classe *CMyObj* drivada da *CObject* e duas classes *CMyObjX* e *CMyObjY* derivadas da *CMyObj*. O objectivo é mostrar um menu popup diferente para um objecto de cada classe. Os menus são criados graficamente no editor de recursos do Microsoft Visual Studio 6.0 como pertencentes a um único menu, como se este se comportasse como um array, com o identificador IDR_CURSOR_MENU. A invocação ou escolha de cada menu é efectuada pela função realçada no código, *GetSubMenu(0)*;

```
CMyObj *pObj;
CMenu menu;
menu.LoadMenu(IDR_CURSOR_MENU);

if(pObj->IsKindOf(RUNTIME_CLASS(CMyObjX))
  menu.GetSubMenu(0)->TrackPopupMenu(
      TPM_LEFTALIGN|TPM_RIGHTBUTTON,
      point.x, point.y, this);
else
if(pObj->IsKindOf(RUNTIME_CLASS(CMyObjY))
  menu.GetSubMenu(1)->TrackPopupMenu(
      TPM_LEFTALIGN|TPM_RIGHTBUTTON,
      point.x, point.y, this);
```

Figura 8. Criação de menus popup específicos de cada objecto.

F. *Barras de ferramentas*

As barras de ferramentas, normalmente situadas por baixo do menu de uma aplicação, têm como objectivo disponibilizar atalhos ao utilizador para as suas opções mais frequentes. Em inúmeras aplicações de software comerciais existe também a possibilidade de personalizar estas barras mediante o perfil de cada utilizador.

Consistem num conjunto de pequenos botões, com um determinado desenho que deve sugerir a função do botão quando este for premido. Cabe ao programador criar estes desenhos, devendo contudo seguir determinadas regras *standard*, como no caso dos menus. As opções mais usuais, como as operações sobre ficheiros, possuem já ícones aos quais o utilizador associa a sua função devido a um certo hábito de utilização e familiarização com as figuras apresentadas. Assim, a criatividade do programador não deverá ter liberdade total devido aos factores acima mencionados. A criação dos desenhos deve seguir uma filosofia de forma a que todos os ícones sejam consistentes. Existem basicamente três esquemas/filosofias de representação:

1. desenhar algo que represente o antes e o depois do comando que se pretende,
2. desenhar a acção do comando (ex. abertura de um ficheiro) e
3. desenhar um objecto que represente a ferramenta necessária para efectuar a operação desejada (ex. impressão de um documento)

Os esquemas mais frequentes são o 2º e o 3º pois levam-nos a um desenho mais simples e a um resultado mais intuitivo. No entanto, não devem ser misturados na mesma barra de ferramentas [3] pois conduzem a uma mais lenta compreensão do significado por parte do utilizador. Contudo, verifica-se que em aplicações comerciais este erro é frequentemente cometido, como se pode ver na fig. 9 onde temos uma barra de ferramentas com ícones pertencentes ao 2º e 3º esquemas apresentados acima.

Figura 9. Barra de ferramentas - Esquema de representação misto (objectos, acções e ferramentas).

Por outro lado, existe também a possiblidade de desenhar os ícones de outra forma, normalmente em barras de ferramentas específicas para uma dada operação. Neste caso é usado o mesmo objecto em vários botões da barra, diferenciados por algo que representa as diferentes acções sobre esse objecto, como é o caso da fig. 10 [4, 5].

Figura 10. Barra de ferramentas – Esquema de representação através de um objecto comum.

VI. CRIAÇÃO, VISUALIZAÇÃO E INTERACÇÃO COM OS OBJECTOS

Nesta secção irão ser mencionadas as várias formas de desenho de objectos no ecrã ou mais concretamente dentro de uma *frame* criada pela aplicação de software que pretendemos projectar. Para desenhar directamente sobre essa janela podem ser utilizados vários dispositivos apontadores, como por exemplo o rato ou outros mais sofisticados e dedicados a criações gráficas. Para mais detalhes acerca de como desenhar directamente no ecrã sugere-se a leitura de [2].

Por outro lado, podemos ter objectos pré-definidos e que podem ser colocados, movidos e/ou editados pelo utilizador na janela da aplicação [4]. Neste caso devemos, como programadores, criar os nossos próprios objectos, utilizando classes derivadas da *CObject*, também pertencente à biblioteca MFC. Cada classe deve possuir uma função para desenhar um dos seus objectos (instâncias da classe) no interior da janela. A biblioteca fornece inúmeras funções para desenho, tais como *Line*, *Rectangle*, *Ellipse*, *Polygon*, etc para figuras geométricas e, por exemplo, *TextOut* para visualizar texto.

De acordo com o modelo documento/vista, devemos associar ao documento os objectos e a sua manipulação, nomeadamente a criação e operações efectuadas sobre eles. A vista deverá ser utilizada apenas para os desenhar e permitir a interacção entre o utilizador e o objecto. A classe da aplicação que implementa a vista, isto é, aquela que é derivada da *CView* (ou *CSrollView*), possui uma função *OnDraw* que deverá ser redefinida para que possamos especificar dentro dela a forma de desenho dos vários objectos. É aqui que serão invocadas as funções de desenho específicas de cada objecto criado e que pertence ao documento.

VII. DIFERENTES TIPOS DE APLICAÇÕES

A programação de uma aplicação de software utilizando o método documento/vista associado à familiarização com a técnica de desenho de objectos no interior de uma janela abre caminho para o projecto dos mais variados tipos de aplicações. De seguida são apresentados três tipos possíveis e os respectivos objectivos.

Dado que é possível dispor de diferentes interfaces para o mesmo documento, poder-se-à criar facilmente uma aplicação para estudo e avaliação de diferentes interfaces por vários perfis de utilizadores. Neste caso a aplicação projectada teria várias "caras" com diferentes características que o utilizador poderia escolher e mudar a qualquer instante. Assim se poderiam efectuar estudos onde se avaliassem diferentes colocações de controlos no ecrã, esquemas de cores, organização dos dados no ecrã pertencentes a uma base de dados, o acesso aos vários comandos e opções da aplicação, etc.

Por outro lado, temos a criação de ferramentas específicas com controlos personalizados e especializados para uma dada área de investigação. Ao nível da simulação de um dado sistema, qualquer que ele seja, podem ser criados graficamente objectos virtuais, ou seja, o mais parecidos possível com os objectos com que lidamos diariamente e interagir com eles. Em sistemas de hardware, por exemplo, poderia ser construída a sua interface através de controlos actuados pelo utilizador, como por exemplo, leds, interruptores, visores, etc.

Finalmente, a personalização de controlos e a utilização de imagens pode conduzir à criação de aplicações com interfaces parecidos com páginas HTML, cuja composição depende apenas da criatividade do programador, com um elevado grau de liberdade, pois os utilizadores da geração actual estão cada vez mais familiarizados com esta tecnologia.

VIII. CONCLUSÕES

As aplicações que utilizam a biblioteca MFC usam normalmente o modelo de programação documento/vista para gerir informação, diferentes formatos de ficheiros e a representação visual dos dados ao utilizador. Na generalidade esta é a forma mais eficiente e apropriada de construir uma aplicação porque separa de forma clara os dados da sua visualização, eliminando código redundante e simplificando assim a sua escrita. No entanto, este não é o método mais apropriado quando se pretende projectar uma aplicação muito simples onde este modelo não se justifique ou se se pretender portar código que à partida misture a gestão dos dados com a sua visualização. Neste caso a separação das duas tarefas pode ser extremamente difícil e por isso este modelo seria indesejável.

Relativamente ao interface da aplicação podemos concluir que embora existam regras e se deva seguir

directivas de standardização, essas regras nem sempre são cumpridas mesmo em aplicações comerciais, como já foi mencionado. Ao projectar uma aplicação o programador tem total liberdade na escolha do interface. No entanto, não deve esquecer o perfil dos utilizadores da sua aplicação e avaliar a sua rapidez de aprendizajem pois este factor é determinante na aceitação e consequente utilização frequente do software produzido [6].

REFERÊNCIAS

[1] Andreia Melo, Beatriz S. Santos, C. Ferreira, J. Sousa Pinto, "Software Application for Data Visualization and Interaction in a Location Routing Problem", Electrónica e Telecomunicações, Vol. 2, Nº 4, pp 471-476, Janeiro, 1999.

[2] Ivor Horton, "Beginning Visual C++ 4", Wrox Press, 1996.

[3] Deborah J. Mayhew, "Principles and Guidelines in Software User Interface Design", Prentice Hall, 1992.

[4] Andreia Melo, "Especificação, Optimização e Teste de Algoritmos de Controlo Hierárquicos", Dissertação de Mestrado em Engenharia Electrónica e de Telecomunicações, Universidade de Aveiro, Janeiro, 2000.

[5] Andreia Melo, Valery Sklyarov, "Ambiente Integrado para Especificação, Projecto e Verificação de Unidades de Controlo em FPGAs", Electrónica e Telecomunicações, Vol. 2, Nº 4, pp 477-485, Janeiro, 1999.

[6] Beatriz S. Santos, J. Sousa Pinto, "An Introductory Course on Human Computer Interaction", Electrónica e Telecomunicações, Vol. 3, Nº 3, pp 228-232, Janeiro, 2001.

Bulldozer: Um robô que reconhece o seu passado

Valter Filipe Silva[1], Frederico Miguel Santos[2]

1 – Escola Superior de Tecnologia de Castelo Branco / DETUA
2 – Instituto Superior de Engenharia de Coimbra / DETUA

Resumo – **Este artigo descreve o robô Bulldozer IV vencedor do concurso Micro-Rato 2001. Trata-se de uma evolução das versões anteriores do Bulldozer, tendo sido adicionado um novo micro-controlador, uma bússola electrónica e um rato óptico de computador. Todo o software e hardware existente foi aproveitado, existindo apenas pequenos ajustes. Serão descritos ao longo do artigo alguns aspectos importantes, tanto na construção do hardware, como do desenvolvimento do software. É dada especial atenção ao cálculo da posição global e regresso ao ponto de partida, sem informação prévia.**

Abstract – **This article describes the Bulldozer IV robot, winner of the Micro-Rato 2001 contest. It is an evolution of previous versions to which a new micro-controller, an electronic compass and an optical computer mouse were added. All the hardware and software of the previous versions of Bulldozer are reused, with just minor changes. In this paper some important hardware and software issues are discussed, particularly global positioning in the maze and returning to initial position without any previous information.**

I. INTRODUÇÃO

O concurso Micro-Rato realiza-se na Universidade de Aveiro desde 1995. A equipa Bulldozer participa no evento desde 1998, tendo efectuado a sua 4ª participação na edição deste ano, o Micro-Rato 2001. Para tornar o concurso mais aliciante, a organização resolveu alterar as regras nesta edição, havendo agora dois objectivos a cumprir.

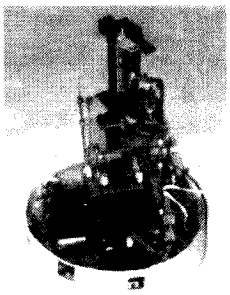


Fig. 1 – O aspecto físico do Bulldozer

O Robô Bulldozer (fig. 1), construído segundo as novas exigências das regras do concurso Micro-Rato, é capaz de atingir a área preta escondida no meio de um labirinto com 5x10m e assinalada por um emissor de infra-vermelhos (1º objectivo), e de seguida regressar ao ponto de partida (2º objectivo) apenas com a informação recolhida no caminho para o 1º objectivo.

Em termos técnicos, o Bulldozer IV reaproveitou todo o hardware e software existente das versões anteriores, tendo sido acrescentado uma bússola electrónica, um rato óptico de computador e um micro-controlador adicional.

II VERSÕES ANTERIORES DO BULLDOZER

O robô Bulldozer participou nas três edições anteriores do concurso Micro-Rato em que os robôs teriam que ser capazes, no interior de um labirinto com 5 x 5 m, encontrar um área de chegada (área preta), assinalada com um farol emissor de infravermelhos, sem colidir com os obstáculos e no menor tempo possível.

As versões anteriores do Bulldozer apenas usavam a placa DET188 com o processador Intel'188 e uma placa de expansão (I/O 188) para fazer o interface com os diversos periféricos (motores, botões, sensores, etc.). Um esquema geral é apresentado na figura 2. O comportamento das versões anteriores é explicado mais à frente neste artigo.

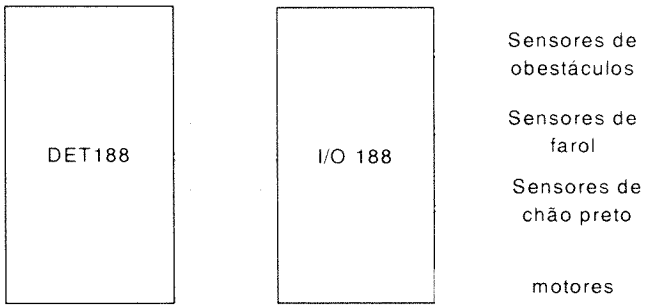


Fig. 2 – O hardware antigo

III. DESCRIÇÃO DO HARDWARE

O hardware do Bulldozer baseia-se, tal como os anteriores, no conjunto DET188 e placa de expansão I/O188, fornecido pela organização [8]. Este conjunto possui um interface directo para 3 sensores de obstáculos, 2 sensores de farol e 1 sensor de área de chegada. A construção de cada sub-sistema do Bulldozer IV é descrita de seguida.

A. *Detecção de Farol*

A detecção de farol é feita através de um sistema de semi-rotação [9], com cerca de 270º de amplitude. A sua posição angular é determinada através de um potenciómetro acoplado ao sistema e ligado a uma ADC. Para o controlo do motor usado na detecção de farol foi construído um circuito electrónico [1] que inverte o sentido de rotação do motor sempre que é atingida uma posição extrema.

Dada as novas dimensões do labirinto (5m x 10m) foram utilizados dois sensores de farol solidários. Um destes sensores colocado a 39cm de altura servia apenas para avistar o farol a longas distâncias (>5m), o outro montado a 28cm de altura permite a detecção precisa da localização do farol (<5m). Esta precisão é aumentada com recurso a um pedaço de tubo preto que limita o ângulo de detecção do sensor.

B. *Detecção de obstáculos*

Relativamente à detecção de obstáculos, as respectivas dimensões mínimas (10 cm) dificultam a detecção por meio de um par simples LED+sensor. Assim, para melhorar a detecção de obstáculos decidiu-se sobre-iluminar as zonas frontal e lateral. Na figura 3 é apresentada a disposição dos LED e sensores utilizados.

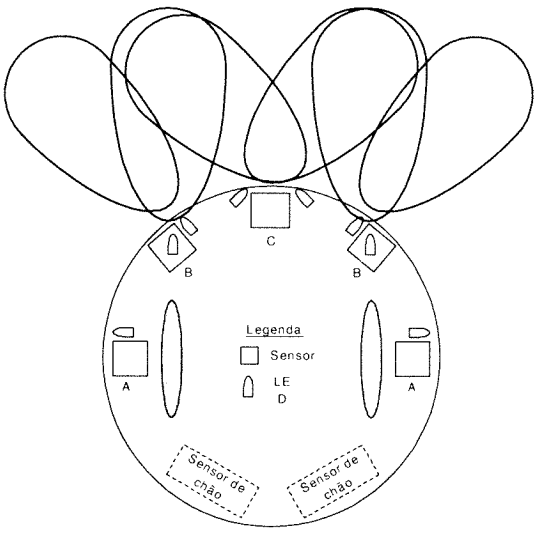


Fig. 3 – Implantação dos sensores de obstáculos

A disposição dos conjuntos B e C melhoram significativamente o comportamento perante situações de mais difícil detecção como por exemplo, esquinas.

C. *Detecção de área preta*

Para a detecção do chão preto foram utilizados dois sensores, colocados simetricamente na parte traseira do robô, para garantir a correcta imobilização na área preta, ver a figura 3.

D. *Sistema de localização*

Para permitir ao Bulldozer o regresso ao ponto de partida, recorreu-se a uma bússola electrónica (Dinsmore 1655) [2], que permite obter em cada instante a direcção do movimento com uma precisão de 1º.

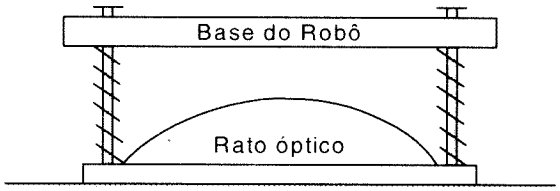


Fig. 4 – fixação do rato óptico

A distância percorrida é obtida com a ajuda de um rato óptico (Microsoft Wheel Mouse Optical), que tem uma precisão de 1/16mm, tendo sido programado para funcionar a uma frequência de 40Hz. Este componente tem diversas vantagens em relação a um odómetro de *encoders*, descrito em [4], entre elas:

- Não existe escorregamento do sistema em relação ao solo.
- Elevada precisão.
- Adaptação mecânica fácil.

O rato óptico foi acoplado ao robô através de molas que garantem um perfeito contacto com o solo necessário para o bom funcionamento do mesmo, ver figura 4.

Para tratar os dados provenientes dos dois periféricos atrás mencionados e para reaproveitar na totalidade o código existente de edições anteriores, foi adicionado um novo micro-controlador (Microchip PIC16F877) [3].

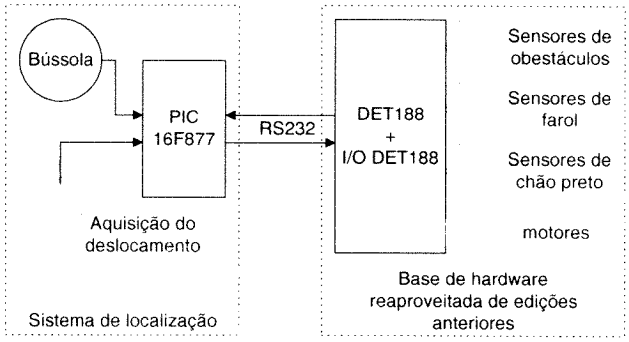


Fig. 5 – Esquema geral do *hardware*

O reaproveitamento de código é feito porque o algoritmo para atingir o primeiro objectivo é o mesmo que para atingir o segundo objectivo. A comunicação entre os dois processadores é feita utilizando RS232 a 9600bps. Um

esquema geral do circuito utilizado é apresentado na figura 5.

E. Bússola

O sensor utilizado tem duas saídas analógicas em forma de seno-coseno, tal como apresentado na figura 5. Estes sinais estão ligados a duas entradas analógicas do PIC.

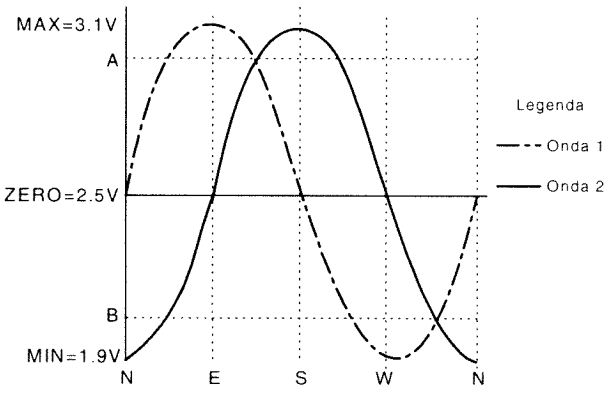


Fig. 6 – Ondas de saída da bússola

Para a obtenção de uma maior precisão utilizam-se apenas as zonas mais lineares das ondas apresentadas, entre os níveis A e B.

O código utilizado para a obtenção do ângulo α (ângulo do robô em relação ao Norte) é o seguinte:

```
if (Onda1 >= A)
    α = acos((Onda2 - ZERO)/(MAX-ZERO));
else
if (Onda2 >= A)
    α = PI-asin((Onda1 - ZERO)/(MAX-ZERO));
else
if (Onda1 <= B)
    α = 2*PI-acos((Onda2 - ZERO)/(MAX-ZERO));
else
if (Onda2 <= B)
{
    α = asin((Onda1 - ZERO)/(MAX-ZERO));
    if (α < 0)
        α = 2*PI + α;
}
```

Fig. 7 – Cálculo de α (em radianos)

Apesar de se ter seguido as indicações do fabricante do componente para o cálculo do ângulo, este apresentava alguns erros, devido a: (1) as ondas não estavam desfasadas de 90º; (2) a tensão pico-a-pico era diferente. Assim, foram feitas correcções por software baseadas em medições práticas.

Para minimizar interferências na bússola foram evitados componentes ferromagnéticos na construção do robô. No caso particular dos motores teve-se cuidado de colocar a bússola suficientemente afastada destes. Dada também a boa blindagem magnética dos motores, a influência causada por estes era praticamente nula.

F. Rato óptico

Este componente baseia-se num circuito integrado HDNS2000 da Agilent [5], que adquire 1500 imagens por segundo, comparando a actual com as últimas para determinar a direcção e a velocidade do movimento.

O rato disponibiliza duas interfaces de comunicação – PS2 e USB, tendo sido utilizada a primeira.

O protocolo PS2 especifica uma comunicação síncrona bidireccional de 8 bits, conforme descrito em [6] e [7].

Para inicializar e configurar o rato é necessário começar por enviar alguns comandos que estão descritos em [6].

A tabela 1 apresenta a trama que é enviada pelo rato com a informação relativa ao último movimento.

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1.	Y_O	X_O	Y_S	X_S	1	M	R	L
2.	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
3.	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0

M	Estado do botão do meio (1=premido)
R	Estado do botão da direita (1=premido)
L	Estado do botão da esquerda (1=premido)
X_0-X_7	Movimento na direcção X
Y_0-Y_7	Movimento na direcção Y
X_S-Y_S	Sentido do movimento (1=negativo)
X_O-Y_O	Overflow (1=ocorreu overflow)

Tabela 1 – Trama de comunicação

Sabendo que a precisão do rato é de 16 incrementos por milímetro e que entre cada trama podem ocorrer 256 contagens sem overflow (em cada sentido), utilizando uma frequência de 40 tramas por segundo, possibilita uma velocidade máxima de 64cm/seg para o robô.

Para se identificar o byte dentro da trama é utilizado um mecanismo de sincronismo, baseado no byte 1 que apresenta o seguinte valor: XXXX1000, já que os botões nunca são premidos.

IV. CÁLCULO DA POSIÇÃO

Para saber a posição do robô em qualquer instante é utilizado um referencial ortogonal, com um dos eixos orientado para o norte magnético. A sua posição de arranque é assumida como (0, 0), havendo duas variáveis que mantêm a sua distância à origem em cada eixo.

Tomando como exemplo a figura 8, e assumindo o deslocamento $\vec{D}$, que é obtido através da informação retirada do rato (componentes A' e B', respectivamente Byte2 e Byte3 da tabela 1), temos no referencial absoluto um deslocamento (A, B).

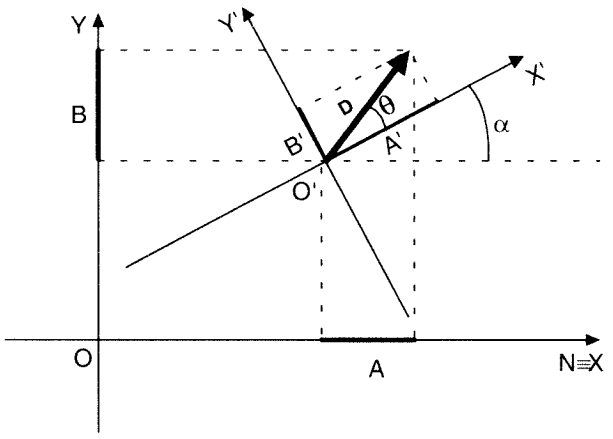


Fig. 8 Cálculo do deslocamento

O referencial O' é o referencial do robô, fazendo um ângulo α com o Norte ($N \equiv X$), ângulo este obtido através da bússola – ver figura 7. Assim, o deslocamento, segundo o nosso referencial absoluto (A e B) é dado pelas seguintes expressões:

$$\theta = \arctan\left(\frac{B'}{A'}\right) \qquad \left\|\vec{D}\right\| = \sqrt{B'^2 + A'^2}$$

$$A = \left\|\vec{D}\right\| \times \cos(\theta + \alpha) \qquad B = \left\|\vec{D}\right\| \times \sin(\theta + \alpha)$$

$$X_i = X_{i-1} + A \qquad Y_i = Y_{i-1} + B$$

Os valores X_i e Y_i representam a posição actual em relação à origem O.

V. SOFTWARE

O software que comanda o Bulldozer, está distribuído por dois processadores (Intel '188 e PIC).

O processador PIC é responsável pela cálculo da localização espacial do robô, enquanto que o Intel '188 é responsável pelas tarefas de controlo.

Toda e qualquer operação que o robô desempenha é controlada por um temporizador especifico, que não deixa o robô executar a mesma tarefa infinitamente. Assim, por exemplo, o robô não anda eternamente a seguir uma parede, ou não anda eternamente a virar-se na mesma direcção à procura do farol. Desta forma o comportamento do robô é melhorado, aumentado a aleatoriedade do sistema, por vezes necessária ao bom desempenho em concurso, para evitar ciclos viciosos.

A. Primeiro Objectivo

Para atingir a área preta (primeiro objectivo) foi utilizado o software para o '188, existente de outras participações (e já com provas dadas!) [1]. Este baseia-se no kernel de tempo real ReTMiK fornecido pela organização e disponível em [8].

O kernel gere a execução de várias tarefas cíclicas, controlando automaticamente a sua activação. Assim, o utilizador apenas tem de especificar o código e frequência de execução (que define também a prioridade) de cada tarefa.

Existem seis tarefas definidas (ordenadas por prioridade):

- STOP – período 20ms, atende os botões de arranque e paragem e é responsável pela imobilização do robô em qualquer situação;
- FAROL – período 20ms, responsável pela leitura dos sensores de farol e respectiva posição angular;
- EVITA OBSTÁCULOS – período 50ms, toma decisões relativamente à direcção a seguir;
- CONTROLO MOTORES – período 50ms, aplica aos motores a direcção obtida na tarefa anterior;
- SENSOR CHÃO PRETO – período 100ms, lê a informação dos sensores de chão preto e decide sobre a correcta posição dentro da área preta;
- RAND – período 3s, esta tarefa gera um valor aleatório.

A tarefa FAROL executa cerca de 30 leituras por varrimento, sendo cada varrimento dividido em 5 partes, conforme a figura 9.

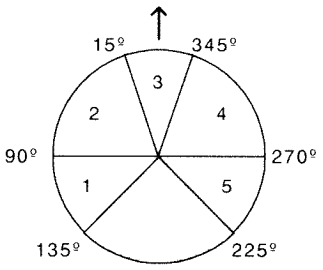


Fig. 9 Divisão do campo de detecção de farol

Para existir uma maior precisão no alinhamento para o farol, a área 3 é a que tem menor ângulo.

O Bulldozer atinge a área preta usando o algoritmo de seguimento de paredes, apresentado na figura 10. Como se pode ver, apenas se inicia o seguimento de paredes quando o farol se encontra por detrás destas, melhorando-se o comportamento do robô, não permitindo que este comece a seguir uma parede desnecessariamente, por exemplo, a parede do labirinto. O fim do contorno é feito quando o obstáculo acaba, ou seja ao ser atingida uma esquina.

O Bulldozer consegue distinguir, em certas situações, quando um obstáculo é uma parede ou outro robô. Isto é possível, quando por exemplo se contorna uma parede pela direita e existe um obstáculo na esquerda. Esta distinção é importante para decidir o que fazer perante tal cenário. Por exemplo, esperar que o obstáculo saia da frente em vez de se desviar deste.

Além deste algoritmo, o robô dispõe de software capaz de reconhecer certas configurações de obstáculos, conseguindo assim situar-se melhor no labirinto e obter uma melhor performance. Através deste método, o Bulldozer resolve algumas configurações de obstáculos propicias a ciclos viciosos. Na figura 11 são apresentadas

duas destas configurações, e que o Bulldozer é capaz de resolver. Nesta figura é apresentada a trajectória mais provável e que leva a uma situação de ciclo vicioso.

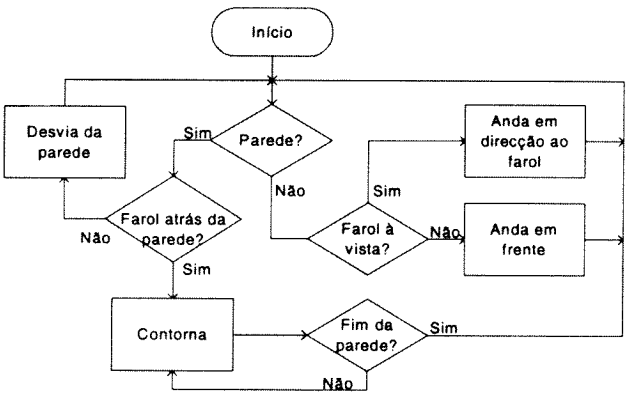


Fig. 10 Algoritmo de contorno de parede

B. *Segundo Objectivo*

Para atingir o segundo objectivo, a novidade no concurso deste ano, foi acrescentado outro processador, responsável pelo cálculo da posição (descrito em IV) e pela gestão de um sistema de faróis virtuais. Este sistema assegura a transparência para o software do DET188 (descrito no ponto anterior) na ida para a área preta, ou no regresso para o ponto de partida. Um farol virtual consiste num ponto no espaço, onde na realidade não existe um farol, mas que o DET188 interpreta como tal. O ponto (0,0), ponto de partida do robô, é um farol virtual e é o objectivo final do robô. Sempre que acaba uma parede é fixado um farol virtual, num máximo de 16, devido às limitações de memória da PIC. No percurso de regresso o robô vai tendo como objectivos os faróis virtuais fixados na ida, mas atingindo-os pela ordem inversa. Assim, e passando por todos os faróis virtuais, consegue-se uma distância mais curta no caminho de regresso, em relação ao caminho percorrido para atingir o primeiro objectivo.

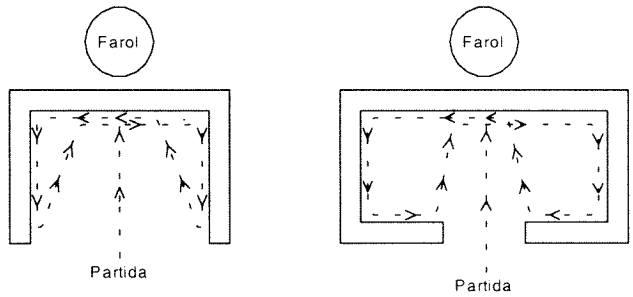


Fig. 11 Duas configurações de obstáculos difíceis

Na figura 12 é mostrado um exemplo de funcionamento do sistema de faróis virtuais. No arranque o robô guarda o farol F_0 (0,0) e tenta atingir o farol real (área preta) Neste percurso são colocados no fim de cada parede os faróis F_1, F_2 e F_3. Ao ser atingida a área preta, o robô desloca-se para a partida tendo que passar por F_3, F_2 e F_1, dirigindo-se de seguida para F_0.

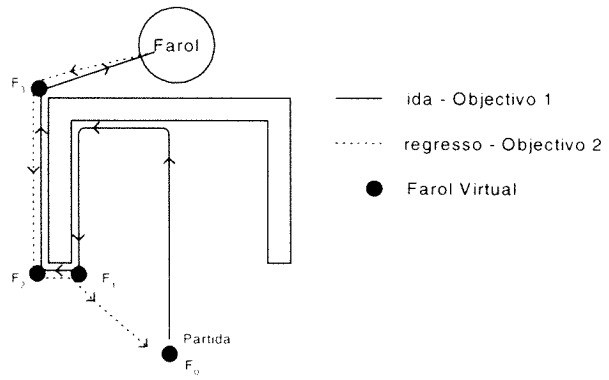


Fig. 12 Exemplo de um percurso

Considera-se que um determinado farol virtual foi atingido quando o robô está dentro de um círculo com raio de ≈0.5m (≈1m para o farol F_0 (0,0)) e a distância do robô ao farol virtual é maior que a calculada no momento anterior. Isto permite que o robô se aproxime o máximo do ponto alvo mesmo para o caso em que o farol virtual se encontra num ponto inacessível (por exemplo, num obstáculo), o que pode ser possível devido a erros acumulados. O algoritmo utilizado para o uso do sistema de faróis virtuais é apresentado na figura 13.

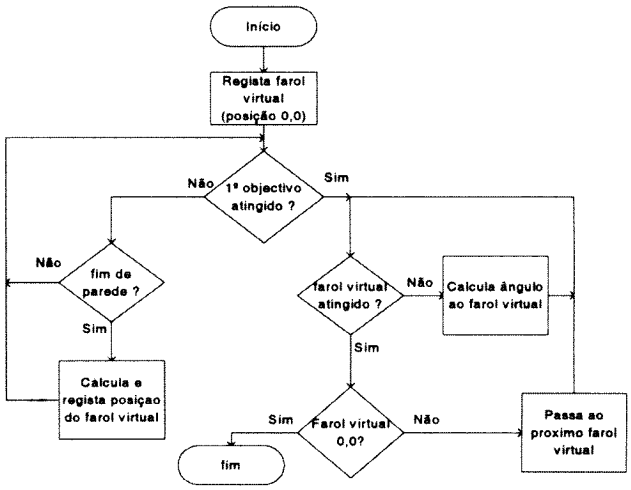


Fig. 13 Algoritmo de aplicação de faróis virtuais

Como já foi referido anteriormente todo o software de gestão da posição do robô e dos faróis virtuais é executado no PIC, tendo este que informar, no regresso, o Intel '188 de qual a direcção angular correspondente (figura 9) em que se encontra o farol virtual. Tomando como exemplo a figura 14, pretende-se saber a direcção angular (β) na qual se encontra o farol virtual, em relação ao norte. Esta direcção é dada por:

$$\beta = \arccos\left(\frac{Xr - Xf}{\left\|\vec{D}\right\|}\right)$$

Através desta formula determina-se o ângulo entre a direcção norte e o vector que une o robô ao farol virtual. Uma vez que a função *arccos* devolve um ângulo entre 0 e π, é necessário fazer uma correcção para o caso do ângulo ser superior a π. Esta correcção é feita com o seguinte código:

```
if (Yr-Yf<0)
    beta=2*PI-beta;
```

O ângulo que a frente do robô faz com o farol virtual é dado por:

$$ang\_farol = \beta - \alpha$$

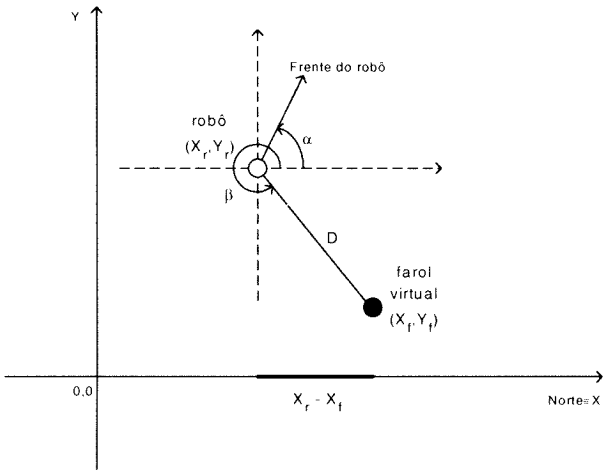


Fig. 14 – Cálculo da direcção do farol virtual

O ângulo calculado (*ang_farol*), permite saber em que direcção angular é que se encontra o farol virtual relativamente ao robô. A informação sobre a área correspondente é enviada para o Intel '188, que a processa como se do farol real se tratasse. O envio desta informação apenas se realiza depois de atingido o primeiro objectivo. Até então o PIC vai sendo informado dos vários eventos ocorridos, por exemplo, o início da prova e o fim de parede para marcação dos faróis virtuais. Quando o segundo objectivo é atingido o '188 é informado, por forma a parar e assinalar este facto.

Um exemplo da comunicação utilizada entre os dois processadores é apresentado na figura 15.

Toda o programação do PIC foi efectuada em C com o compilador da Ht-soft [11]. Em particular, as funções trigonométricas fazem parte da respectiva livraria matemática.

VI. RESULTADOS

A disposição dos LEDs/sensores de obstáculos e chão preto revelou-se eficaz ao longo dos 4 anos de participações, fazendo com que o Bulldozer evite os obstáculos mais problemáticos, e se consiga imobilizar na área preta sem sofrer penalizações.

A detecção de farol, também utilizado desde 1998, é muito precisa, de tal forma que o robô dirige-se para o farol sempre orientado para este.

O sistema adoptado para retornar ao ponto de partida revelou-se eficaz e de construção muito simples.

Nesta edição, o Bulldozer foi o único robô capaz de atingir o ponto de partida, segundo as novas exigências das regras do concurso.

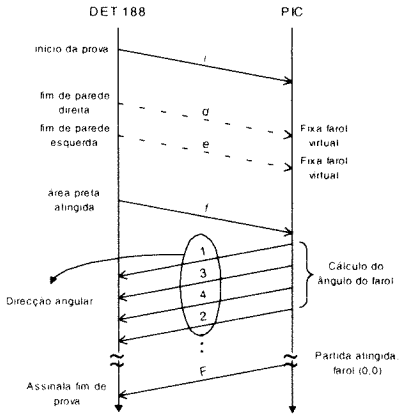


Fig. 15 – Comunicação entre os processadores

Nas quatros mangas disputadas pelo robô este conseguiu sempre atingir o farol e, em duas delas, voltou ao ponto de partida. Numa destas duas vezes o robô imobilizou-se exactamente no ponto de partida, enquanto na segunda ficou a cerca de 20 cm deste, depois de ter percorrido mais de 25m.

Nos testes efectuados antes do concurso o robô conseguiu sempre regressar ao ponto de partida, com erros de cerca de 2%, não o conseguindo em todas as mangas do concurso devido à sua velocidade baixa de locomoção e limitação do tempo de prova.

O desenvolvimento deste sistema só foi possível porque a equipa já tinha o sistema de atingir o primeiro objectivo completamente desenvolvido e testado.

VII. CONCLUSÕES

O software do Intel'188, foi utilizado pela primeira vez no Bulldozer 98, tendo obtido o 1º lugar, e desde então sofreu apenas ligeiros ajustes. Este tem revelado uma excelente *performance* para as exigências do concurso ao longo destes anos. Este software permitiu que nas 15 mangas disputadas pelo Bulldozer (em 4 participações), apenas não terminasse uma manga, permitindo ao robô uma elevada percentagem de sucesso.

Na edição de 2001, com as novas exigências do concurso, o Bulldozer, foi o único robô a cumprir integralmente os objectivos propostos pela organização. A bússola electrónica e o rato óptico revelaram-se determinantes para a obtenção de tal feito. O uso destes dois componentes juntamente com uma arquitectura

multiprocessador valeram ao Bulldozer o prémio inovação e o primeiro lugar absoluto.

O palmarés do Bulldozer nas quatro participações é indicado na tabela 2.

Edição	Competição	Prémios de Mérito
1998	**1º**	Engenharia e DETUA
1999	4º	
2000	2º	DETUA
2001	**1º**	Inovação

Tabela 2 - Palmarés

Todas as informações relativas ao Bulldozer podem ser consultadas em http://bulldozer.4mg.com.

VIII. AGRADECIMENTOS

A equipa Bulldozer agradece a todas as pessoas ou instituições que de forma directa ou indirecta contribuíram para o sucesso da equipa nas várias participações.

REFERÊNCIAS

[1] Valter Silva, Telmo Silva, Frederico Santos, "Bulldozer: O Valor da Aleatoriedade", Electrónica e Telecomunicações, Vol. 2, Nº 6, pp. 804-805, 1999.

[2] "Analog Hall-Effect sensor; part no. 1655", Dinsmore Instrument Company, 1998.

[3] "PIC 16F87x Data Sheet", Microchip, 1999.

[4] Edwin Wise, "Applied Robotics", Prompt Publications, 1999.

[5] "HDSN2000 Technical Data", Agilent Technologies Inc., 2000.

[6] "Synaptics TouchPad Interfacing Guide", Synaptics Inc., 1998.

[7] Tomi Engdahl, "PC Mouse Information", disponível em: http://digitais.ist.utl.pt/ec-asc/proj2/mouse.html.

[8] Concurso Micro-Rato, Departamento de Electrónica e Telecomunicações da Universidade de Aveiro, http://microrato.ua.pt/

[9] João Carlos Capucho, José António Parente, "Detecção do Farol: Estudo para Aplicação no Robot *Tripé II*", Vol 2, Nº 6, pp. 806-807, 1999.

[10] J. Borenstein, H.R. Everett, L. Feng, "Where am I? Sensors and Methods for Mobile Robot Positioning", University of Michigan, April 1996, disponível em http://www.eecs.umich.edu/~johannb/pos96rep.pdf

[11] Ht-Soft Web site, www.htsoft.com

Approximating linear time with finite count clocks

Pedro Fonseca

Abstract **– In computer systems, actions are triggered, not only by external events, but also by the passing of time. In the case of real-time systems, these actions must be executed under strict time constraints. Deciding on what actions to take depends on the correct result of operations like comparing time instants. These are complicated by the fact that clock counters have a limited life span. If care is not taken, their wrap-around may present problems and limit the operating life of a system. We present a simple method to circumvent this problem, based on a counter that has a life span of twice the larger interval of interest. This method is easily and efficiently implemented by using the type cast facilities of languages such as C.**

I. TIME AND CLOCKS IN COMPUTER SYSTEMS

Computer systems interact with their environment. In many cases, the start and stop of computer actions are triggered, not by any user interaction, but rather by the passing of time. To execute an action constrained by time, computer systems must contain a device to perceive time: an internal clock. This clock is used by the computer system to take decisions to start and stop the execution of tasks, in order to respect its time constraints. We find here two notions of time. On one side, there is the time that controls the evolution of events in the computer environment. This time is external to the computer system and corresponds to the physical quantity: we call it *physical time*. On the other side ("at the same time", one could say) there is time as perceived by the computer system. This is the time that commands the start and stop of the computer internal actions. We call this *clock time*.

Physical time is (at least, for all practical purposes...) a linear time. We can represent it by a straight line (fig. 1), and events are points in that line. The line is infinite. In our perception of the Universe, time has always existed and it will continue to exist. The notion of an instant (or a time...) after which there would be no more time makes no sense to our minds. Every point in the time line is reached once and only once. Before we have reached that instant, it belongs to the future; for one instant, it is present; after we have reached it, it will be past and it will be no longer reachable (an idea with tremendous implications in our life and psychology).

Dep. de Electrónica, Universidade de Aveiro, P3810-193 AVEIRO, Portugal, E-mail: pf@det.ua.pt, Tel. +351 234 370984, Fax: +351 234 381128, URL: http://sweet.ua.pt/~pf

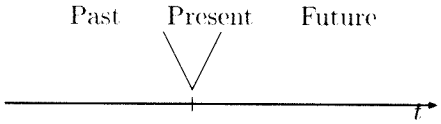


Figure 1 - Linear time

We generally consider time to progress from left to right in a horizontal line. Taking some point to be the present, the past is on the left side, the future is on the right (no political consequences should be inferred from the preceding statement...). We use notions like *happened before* or *happened after* since tender age. We can assign a time mark to every event, and these can be used to establish a sequence of events. We define an ordering of events using only their time marks: the sequence of events corresponds to the sequence of time marks.

All clocks in a computer system are based in the same principle: an oscillator produces periodic events, the clock *ticks*, and a counter is incremented at each clock tick. The value stored in the clock is clock time. Because it is based on a counter, clock time differs from physical time in two fundamental aspects. First, clock time is discrete, whereas physical time is continuous. In mathematical terms, measuring time with a clock is an application C, defined by $C : \mathbb{R} \rightarrow \mathcal{V}$, where $\mathcal{V}$ is a discrete set. We represent physical time by the letter t, and clock time by the letter c. $c = C(t)$ is the clock value that corresponds to physical time t. For sake of simplicity, and w.l.o.g., we will consider that physical time and clock time are measured using the same units and that clocks are perfect (there is just a quantization error and no change of scale from physical time to clock time).

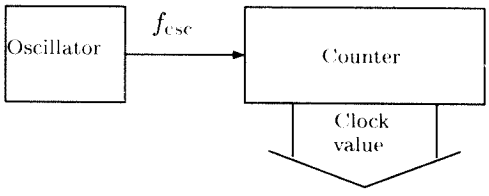


Figure 2 - Model of physical clock

The second difference is that, whereas physical time is infinite and unbounded (at least, to our perception), possible values for clock time are finite and bounded. In mathematical terms, $\mathcal{V}$ is a finite set. We call $M = \#\mathcal{V}$ the clock counter modulo. In many cases $\mathcal{V} = \{0, 1, 2, \ldots, M-1\}$. The count starts at 0 and,

when it reaches $M-1$, the clock value returns to the initial value at the next count, restarting the cycle. This means that clock time is a *circular* time (fig. 3): any point in a circular time is reached repeatedly, over and over (given enough time for the counter to wrap around). Unless the oscillator is stopped, this will go on forever and ever (once again, our notion of never ending time...). This corresponds to a notion of a time represented, not by a straight line, but by a circle. A circular time presents a point of singularity: the clock's value increases as time elapses except in the point when the counter overflows. At this point, a sudden decrease of size $M-1$ occurs.

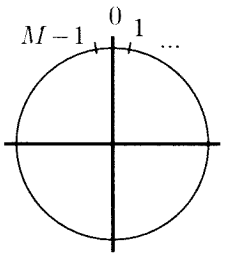


Figure 3 - Circular time

Common uses of clocks in a computer system are to measure the time elapsed between two instants, to determine whether some time instant belongs to the past or to the future or to know how much time we have left before a given instant. These can be generally referred to as *comparing* time instants.

The fact that V is a finite set (and that, by this reason, clock time is a circular time) introduces some significant differences between clock time and physical time. First, all computations in clock time are performed in modulo M. Secondly, whilst in physical time, the order of events can be unambiguously derived from the instants of their occurrence, the same is no longer true in clock time. A smaller clock value can correspond to a later instant. Moreover, given two clock values $c_1 = C(t_1)$ and $c_2 = C(t_2)$, even if we know their order of occurrence, there is an infinite number of possible values for the difference between them. We want to measure the amount of physical time elapsed between two instants, given the respective clock time. Every value in the set

$$\mathcal{S} = \{x : x = c_2 - c_1 + kM, k \in \mathbb{Z}\} \qquad (1)$$

is a possible solution for the time that elapsed from c_1 to c_2. Obviously, only one of them is correct. But, without further information, we have no means to identify the correct solution.

II. The assumptions

The previous section presented some differences between physical time and clock time and how these cause some ambiguities in the comparison of clock time instants. Namely, we have seen that when trying to compute the time elapsed between two instants, we find an infinite number of solutions and that, without some further information, we are unable to identify the correct solution. In this section, we will present the assumptions that will remove these ambiguities.

In order to do this, we introduce the concept of *interval of interest*. This means an upper bound on the difference between two (physical) time instants that we want to compare, using the corresponding clock time values.

This is used in the assumption:

Assumption 1: The length of any interval of interest is smaller than half the modulo of the counter.

This guarantees that our clock will not complete a full turn during any interval we want to measure. Moreover, it allows us to choose, between the set of possible solutions, the one that corresponds to the smallest arc in the circle of time. In numerical terms, it corresponds to the solution with the smallest absolute value.

Lemma 1: The correct solution for the difference between two clock time values, amongst all numerically possible solutions, is the one with the smaller absolute value.

Any other solution would span over an interval that is larger that half the counter modulo, thus contradicting assumption 1. This means also that values of k in eq. (1) are restricted to the set $\{-1, 0, 1\}$. Note, by the way, that, although c_1 and c_2 belong to V, the result does not necessarily belong to V.

III. Sign conventions

Many programming languages, such as C, allow us to define the value stored in a register (which can be a counter) as signed or unsigned. This distinction goes down to the microcode level: most microprocessors distinguish between signed and unsigned operations in their instruction set. In a 8-bit register (a char in C language), the modulo M is $M = 2^8 = 256$ and every value is in the range $\{0, 1, ..., 255\}$. This is the convention for unsigned char values. For signed quantities, the representation is in 2's-complement. The values from 80_h[1] to FF_h represent negative values: 80_h represents -128, FF_h represent -1. If we consider a clock counter with 8 bits, an unsigned char counter would display values from 0 to 255 and then 0 again, and so on. A signed one would display values from -128 to 127, then -128 again, and so on. But, although the significance we assign to the stored values is different in the signed and unsigned case, the actual bit values in the counter are identical. The difference between signed and unsigned is a mere convention.

The problems with comparing clock time values occur when the singularity point lies in the interval between the two values (*i.e.*, when the interval of interest contains the singularity point). Where does this singularity happen? If the counter value is unsigned, it will happen when the count reaches 255 (in a 8-bit

[1] We will use hexadecimal notation to represent the bit pattern stored in the counter register, and decimal notation to represent the value as we read it.

counter): the next value will be 0. But, to a signed value counter, the same transition poses no problem. It corresponds to the counter going from -1 to 0. The same applies to the counting in the antipodes. A signed counter will wrap around in the transition from 127 to -128. For the unsigned counter, this is just going from 127 to 128. All clear, all subtractions work fine. Note that the singularity points for each value convention are half the counter modulo apart.

As we have seen before, what's physically in the counter is independent of whether a counter is signed or unsigned. That is just related how we convene to use the count to represent a value. Second, the same increment can be awkward in one representation (the count decreases 255 when it should increment 1) and totally normal in the other.

So, to avoid the problems caused by counter wrap-around, we just need to choose the representation that eliminates the singularity in the count. Having restricted the larger interval to less than half the maximum count, we know that it can cross the singularity point for one, and only one, of the value conventions.

IV. The algorithm

To implement the algorithm, we consider the full range of the counter divided into four quadrants, 1 to 4 (corresponding to an unsigned count) (fig. 4). The singularity points are in the frontier of quadrants 4 and 1 (for unsigned counting) and of 2 and 3 (for signed counting).

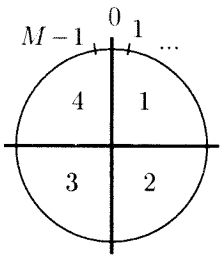


Figure 4 - 4 quadrants

Lemma 2: Comparisons between values in the same quadrant pose no problem.

Lemma 3: Comparisons between values in quadrants 1 and 2 or 3 and 4 pose no problem.

We are left with the problem of comparing values in quadrants (4,1) and (2,3). It is now obvious that:

Lemma 4: Comparisons between values in quadrant 4 with values in quadrant 1 (or *vice-versa*) yield correct values if signed values are used.

Lemma 5: Comparisons between values in quadrant 2 with values in quadrant 3 (or *vice-versa*) yield correct values if unsigned values are used.

To implement the algorithm, we arbitrarily choose a default behaviour (signed or unsigned). The algorithms just needs to check if the values are in the range that contains the singularity point. If this is the case, the alternative behaviour is selected. Otherwise, the computations are performed in the default behaviour. Figure 5 presents the algorithm. In this case, the default behaviour is unsigned. The singularity is in the frontier of the 1st and 4th quadrant; if the values fall into this range, the signed behaviour is selected.

{Default behaviour is unsigned}.
{y is a signed value}
if $x_1 \in Q_1 \cup Q_4$ and $x_2 \in Q_1 \cup Q_4$ **then** {compute using signed values}
  $y = (\text{signed})x_1 - (\text{signed})x_2$
else {compute using unsigned value}
  $y = (\text{unsigned})x_1 - (\text{unsigned})x_2$
end if

Figure 5 - Algorithm for comparing clock values

An implementation is presented in fig. 6, using a char type counter as example. The default for char values is signed. The extension for counter sizes larger than the char type should be adapted to the target processor, depending on the word size in the processor, memory and data bus. Note that, for large sized counters, like 32-bit wide and more, we do not need to use the whole counter for comparisons. Starting with the most significant byte (or word, or ...), a bound-and-branch approach will significantly reduce the comparison time.

```
/*
  compare(x1,x2)

  Calculates the difference between x1 and x2
*/
char compare(char x1,char x2)
{
  char y;

  /* Test if x1 and x2 are in the upper quadrants */
  if( (x1> (signed char)0xC0 || x1 < 0x40) && \
      (x2> (signed char)0xC0 || x2 < 0x40)){
    /* Calculate using signed values */
    y  = x1 - x2;
  }
  else{
    /* Default behaviour */
    y = (signed)((unsigned)x1-(unsigned)x2);
  }

  return y;
}
```

Figure 6 - Example in C code

V. Conclusions

Comparing time instants is a frequent operation in real-time systems, which is some times made difficult by the fact that clock counters have a finite life span. We have presented a method to perform comparisons of time instants using clock values that circumvents these difficulties. The method is based in providing the system with a clock that has a life span larger that twice the maximum interval of interest. The algorithm and an implementation in C language were presented.

Audio Compression: Discussion of an Alternative Approach

João Manuel Rodrigues, Ana Maria Tomé

Abstract – **In this paper we present and dicuss a new algorithm for the compression of audio signals in which we are currently working. Backward-adaptive quantization, a fundamental innovation that differentiates this from other perceptual audio coders, is justified for its multiple advantages. We conclude with a discussion of some issues related to the design of the various components of the coder we are developing.**

Resumo – **Neste artigo expomos e discutimos um novo algoritmo de compressão de sinais áudio em que estamos a trabalhar. A utilização de quantização retro-adaptada, uma inovação fundamental que diferencia este de outros codificadores perceptuais de áudio, é justificada pelas suas múltiplas vantagens. Concluímos com uma discussão de algumas questões relacionadas com o projecto dos vários componentes do codificador que estamos a desenvolver.**

I. INTRODUCTION

Since the introduction of the Compact Disc, digital coding of audio signals has become a common and popular technology. The simple 16-bit linear PCM format used in CDs, however, is now regarded as a very inefficient representation of audio content with its bit rate of 706 kbit/s per channel. As a consequence, new coding algorithms have been developed that can "compress" the audio information into a fraction of the bit rate with little or no degradation in perceived quality. This new generation of coding systems owes its great efficiency to the use of *perceptual coding* principles, i.e.: coding the signal in such a way that the injected noise is rendered inaudible by exploiting the limitations of the auditory system [1]. Since relevant psychoacoustic phenomena are highly dependent on the spectral content of the signal, it is not surprising that most high-quality digital audio coders such as the standard MPEG-Audio Layers I, II, III [2], and AAC [3], are based on sub-band or transform coding techniques. These coders share a common generic structure: a multirate filter bank or a lapped transform splits the input signal into subsampled frequency bands; a psychoacoustic model dynamically estimates the amount of noise that can be added to each band while still being masked by the signal itself; this *masking threshold* as well as bit rate constraints are then used to compute new step sizes and bit allocation for the sub-band quantizers; finally, entropy coding and multiplexing of the sub-band samples, step sizes and allocation information generates the output bit stream. At the other end of the communication channel the bit stream is parsed and demultiplexed, and the quantized sub-band samples are recovered. The sub-band sequences are then combined by an inverse transform or filter bank to produce the output signal.

In this paper, we present a perceptual audio coder with an alternative structure, shown in figure 1. It is based on the same frequency-domain coding principle, but differs from others in essentially one respect: the adaptation of the quantizers is derived, under perceptual considerations, not from the original signal but from previously quantized samples. That is: the system is backward-adaptive. Since this is not a common approach, we devote the next section to address the advantages and summarize some results that support the use of backward adaptation. We then discuss proposals for the implementation of the key components of the system: the filter bank, the perceptual adaptation algorithm, the quantization and entropy coding.

II. BACKWARD ADAPTATION IN PERCEPTUAL CODING

All common perceptual audio coders use a forward-adaptive quantization scheme. Access to the uncorrupted input signal can, in theory, lead to a more accurate adaptation. However, the need of embedding adaptation parameters (either bit allocation and/or quantizer scale factors) in the transmitted information will, in practice, compromise this advantage. The problem is that in order to reduce the amount of this side information, it must be quantized and decimated, thereby reducing its original accuracy.

In the proposed system, on the contrary, the quantizer adaptation parameters (Δ in figure 1) are derived through perceptual and bit rate considerations from the previously quantized signal. The obvious advantage of this backward adaptation scheme is that no side information must be transmitted since the decoder replicates the encoder procedure to reproduce the adaptation parameters. Freed from the constraints imposed by limited channel capacity, there is no need to reduce the time, frequency, or magnitude resolution of the adaptation information, so it can evolve smoothly sample-by-sample. Algorithm design and implementation are much simplified because there is no side information to quantize, encode and multiplex. This should be a significant advantage since the optimal selection of adaptation parameters in an advanced forward-adaptive coder is not a trivial matter [4].

There are, of course, disadvantages in pure backward-adaptive systems. First of all, computational requirements of the decoder are increased by the inclusion of the adaptation algorithm. A related problem is that any future improvements in psychoacoustic modeling cannot simply be integrated into an encoder but must be included in every decoder too. In multimedia applications, programmable devices are the norm and downloading program updates is quite usual. Even in cheap, portable solid-state music players, there is a current trend towards programmable, multifunction devices [5], so this does not seem to be a major difficulty. Still, these problems can be mitigated, and some

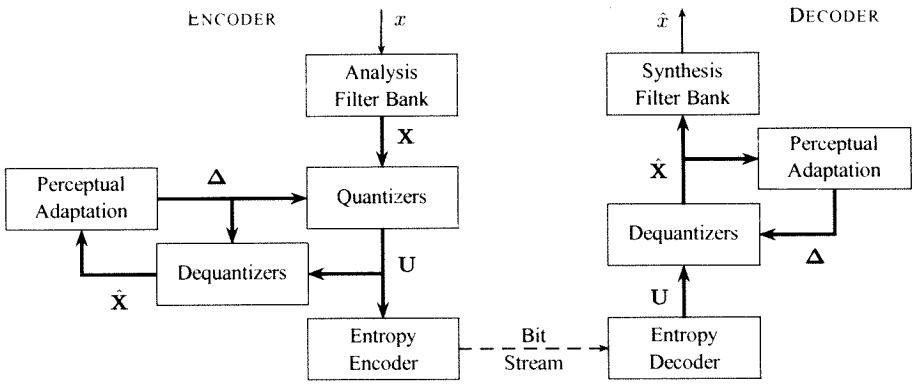


Figure 1 - A frequency-domain perceptual coder with backward-adaptive quantization.

versatility regained by transmitting small amounts of side information, in a hybrid backward- and forward-adaptive fashion. This has also been recognized and implemented by the designers of AC-3 [6], which can be considered a backward-adaptive system, although not quite to the same extent as ours.[1]

Another potential problem of a different, more fundamental nature, is the possibility that the noise introduced by the quantization may disturb the process leading to grossly inaccurate adaptation and poor performance. This could be a serious drawback because, at high compression ratios, large amounts of noise are introduced, and furthermore, this could be aggravated by the nonlinear psychoacoustic computations included in the adaptation loop. In fact, results from previous work with a backward-adaptive coder show that the effect of quantization noise on the backward estimation of psychoacoustic parameters is very small even when very coarse quantization is employed [7]. Also, the performance of that coder was favorably compared to an idealized forward-adaptive version [8].

III. Design Issues

In this section, we discuss some coder design issues, taking into account that the option for a backward-adaptive scheme conditions every component of the system.

A. The Filter Bank

The filter bank is used to decompose the input into several subsampled band-limited signals. There are fundamentally two reasons for doing this. Firstly, it is well known that audio signals are highly correlated and that sub-band coding is an effective means to exploit that redundancy [9]. The other reason is that it provides a direct way to control the spectral shaping of the introduced noise, which is convenient to take advantage of the masking properties of the ear.

The filter bank is crucial to the overall performance of the coder, and it should satisfy several requirements such as critical subsampling, perfect reconstruction, and temporal and spectral resolution compatible with auditory masking constraints [10]. A modulated lapped transform (MLT) [11], which is a particular type of multirate filter bank, is a particularly attractive option because it meets the first two requirements with relatively low delay, and allows fast implementations. It is not surprising that the MLT, also known as the modified discrete cosine transform (MDCT) or time-domain aliasing cancellation (TDAC) filter bank, is one of the most popular transforms in audio coding. The MLT, however, decomposes the signal into equally subsampled, equal bandwidth channels, while the time and frequency resolution of the ear is known to vary widely: at low frequencies, the critical bands (a measure of the frequency selectivity of the ear) are narrow and temporal masking lasts longer; at high frequencies the opposite occurs. Nonuniform decompositions, generally based on tree structures, may be used to approach these multiresolution properties, but can give rise to higher complexity and delay, as well as difficulties in getting proper frequency responses. Another, more common, approach is window-switching, that is: commuting between high and low dimension transforms to trade between temporal and spectral resolution when needed. This is simpler but still it requires a transient detection algorithm and additional constraints on the design of the transform windows.

We believe that in our samplewise adaptive coder the resolution constraints are not so stringent, and a simple fixed-dimension MLT may very well be effective. Assuming 44100 samples/s signals, the frequency resolution of a 256-band MLT, for instance, will be 86 Hz, which is narrower than the lower critical bands, and, at 5.8 ms, the time resolution is certainly good enough to avoid violating post-masking thresholds even in high frequency bands. This contrasts with forward-adaptive systems such as MPEG Layer I where, despite much coarser frequency resolution with only 32 bands, quantizer adaptation only occurs once every 8.7 ms.

B. Quantization and Coding

In backward-adaptive coders, it is very important that quantizers support a large dynamic range because there is no advance information about sudden attacks in the signal. On the other hand, it is psychoacoustically acceptable to introduce larger absolute errors in larger amplitude signals. Therefore, nonuniform quantization with a near-logarithmic companding rule seems quite appropriate. It is also essential to use mid-tread quantizers because their signal-to-noise ratio is never below 0 dB, but also because

[1] In fact, the bit allocation algorithm in AC-3 uses only a fraction of the transmitted information, resulting in coarser adaptation.

a mid-rise quantizer could eventually make the adaptation loop turn unstable. Two degrees of freedom can easily be controlled in a logarithmic quantizer, which determine the maximum absolute error introduced in small signals and the maximum relative error in large signals. The adaptation algorithm can manipulate either or both of these parameters for each quantizer.

The quantized outputs are coded using an arithmetic code [12]. Arithmetic coding is the most efficient form of entropy coding known, not being restricted to encode each symbol with an integral number of bits like Huffman codes. Furthermore, it promotes a clear separation between coding and statistical source modeling, which allows an easy integration of reasonably complex, highly dynamic, context-adaptive source models. Some preliminary measurements confirm that the statistical distributions effectively vary both in time and from band to band, so adaptive models are advisable.

C. Perceptual Adaptation Algorithm

Auditory models found in the perceptual audio coding literature are traditionally based on the concept of masking threshold, i.e., they dynamically estimate the amount of noise that may be added to a signal without causing audible distortion. In spite of its appeal, it is recognized that this concept, or at least its simplistic interpretation, suffers from several problems [13]. Another approach to auditory modeling is to quantify the ability of the ear to perceive differences between two signals—e.g. the input and output of a coding system—by comparing some internal representation of these signals. This strategy has been fruitfully applied in audio quality measurement but not in audio coding. The reason for this is certainly the higher complexity that it involves. However, for any given coder, there is some a priori knowledge of the kind of distortion that will be introduced. Therefore, an internal representation model for audio coding may not need the full generality of those used in quality measures. We are currently exploring the applicability of this approach to audio coding. The studied model computes the excitation pattern from the spectral representation obtained by the coder transform, and it is relatively simple but plausible since it is based on a recent audio quality measure [14]. An approximate formulation has been derived that permits the evaluation of the effect of quantization noise on the excitation patterns predicted by this model. This derivation and some preliminary validation results will be presented in [15].

IV. CONCLUSIONS

Several high-quality audio coding algorithms are in widespread use nowadays, but all follow a similar strategy. This paper proposes an alternative approach to audio compression in which backward adaptation plays a significant role. We discussed the implications of this option and laid out research directions for the development of the system. Previous results with a preliminary version show the viability of the approach, and we expect to achieve very good performance with a low complexity.

REFERENCES

[1] Peter Noll, "Wideband speech and audio coding", *IEEE Communications Magazine*, pp. 34–44, Nov. 1993.

[2] Karlheinz Brandenburg, Gerhard Stoll, et al., "The ISO/MPEG-Audio codec: A generic standard for coding of high quality digital audio", in *92nd AES-Convention*, Vienna, Mar. 1992, Audio Engineering Society, Preprint 3336.

[3] Marina Bosi, Karlheinz Brandenburg, Schuyler Quackenbush, Louis Fielder, Kenzo Akagiri, Hendrik Fuchs, Martin Dietz, Jürgen Herre, Grant Davidson, and Yoshiaki Oikawa, "ISO/IEC MPEG-2 advanced audio coding", *Journal of the Audio Engineering Society*, vol. 45, no. 10, pp. 789–813, Oct. 1997.

[4] Ashish Aggarwal, Shankar L. Regunathan, and Kenneth Rose, "Near-optimal selection of encoding parameters for audio coding", in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Salt Lake City, May 2001.

[5] Jason Kridner, Mark Nadeski, and Pedro Gelabert, "A DSP powered solid state audio system", in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Mar. 1999, pp. 2283–2286.

[6] Craig C. Todd, Grant A. Davidson, Mark F. Davis, Louis D. Fielder, Brian D. Link, and Steve Vernon, "AC-3: Flexible perceptual coding for audio transmission and storage", in *96th AES-Convention*, Audio Engineering Society, Feb. 1994, Preprint 3796, (also in http://www.dolby.com/tech/ac3flex.html).

[7] João Manuel Rodrigues and Ana Maria Tomé, "A backward-adaptive perceptual audio coder", in *EUSIPCO*, Trieste, Italy, Sept. 1996, vol. II, pp. 1007–1010.

[8] João Manuel Rodrigues and Ana Maria Tomé, "On the use of backward adaptation in a perceptual audio coder", *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 4, pp. 488–490, July 2000.

[9] Nuggehally S. Jayant and Peter Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video*, Prentice-Hall, Englewood Cliffs, N.J., 1984.

[10] James D. Johnston and Karlheinz Brandenburg, "Wideband coding—perceptual considerations for speech and music", in *Advances in Speech Signal Processing*, Sadaoki Furui and M. Mohan Sondhi, Eds., chapter 4, Marcel Dekker, Inc., New York, 1991.

[11] Henrique S. Malvar, *Signal Processing with Lapped Transforms*, Artech House, Norwood, MA, 1992.

[12] Ian H. Witten, Radford M. Neal, and John G. Cleary, "Arithmetic coding for data compression", *Communications of the Association for Computing Machinery*, vol. 30, no. 6, pp. 520–540, June 1987.

[13] Raymond N. J. Veldhuis, "Bit rates in audio source coding", *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 1, pp. 86–96, Jan. 1992.

[14] Thilo Thiede, William C. Treurniet, Roland Bitto, Christian Schmidmer, Thomas Sporer, John G. Beerends, Catherine Colomes, Michael Keyhl, Gerhard Stoll, Karlheinz Brandenburg, and Bernhard Feiten, "PEAQ—the ITU standard for objective measurement of perceived audio quality", *Journal of the Audio Engineering Society*, vol. 48, no. 1/2, pp. 3–29, Jan. 2000.

[15] João Manuel Rodrigues, Ana Maria Tomé, and Tomás Oliveira e Silva, "Auditory models in audio coding", in *111th AES-Convention*, New York, Sept. 2001, Audio Engineering Society, (To be presented.).

ARTES GRÁFICAS
AG
Sérgio Cabaço