O METODO JSD COMO APOIO à PROGRAMAÇAO MODULAR E ORIENTADA A OBJETOS

ANA LUZIA GONÇALVES SARINO (*)

RESUMO:

Utiliza-se o método de desenvolvimento de software denominado Jackson System Design - JSD - para especificar um sistema a ser implementado em Modula-2, utilizando-se o conceito de programação modular e orientada a objetos. Um mapeamento entre JSD e a abordagem orientada a objetos é realizado. Os resultados desse experimento são relatados, entre os quais a adequação de JSD para esse tipo de abordagem.

ABSTRACT:

Jackson System Design - JSD - is utilized to specify and design a system to be implemented in Modula-2 following the rules determined by the modular and object oriented approach. A mapping from JSD to the object oriented approach is presented. The results obtained are reported. One of these conclusions is the adequacy of JSD to a modular and object oriented approach.

(*)
Bacharelanda em Ciências de Computação, no Instituto de Ciências Matemáticas de São Carlos, USP. End. ICMSC-USP, Cx. Postal 668, São Carlos, SP.

Este trabalho contou com o apoio financeiro da FAPESP.

O METODO JSD COMO APOIO à PROGRAMAÇAO MODULAR E ORIENTADA A OBJETOS

1. INTRODUÇAO

O termo Engenharia de Software envolve grande número de disciplinas, que de um modo geral procuram, entre outros objetivos, aumentar a qualidade e confiabilidade do software, aumentar a produtividade do processo de desenvolvimento de software e permitir manutenção mais eficiente. O método para análise e desenvolvimento do software e a linguagem utilizados têm um papél importante no alcance dos objetivos declarados acima.

As linguagens de programação têm evoluido para dar apoio ao desenvolvimento de programas (sistemas) de grande porte. Dois recursos importantes para esse fim são: a facilidade de criação de módulos independentes e os tipos abstratos de dados. ADA e MODULA-2 são dois exemplos de linguagens que oferecem esses recursos.

Com o uso de tais linguagens a interface entre partes distintas de um sistema pode ser definida precisamente, de tal forma que componentes produzidos por diferentes programadores sejam compativeis. Além disso, os problemas originados por modificações incorretas em dados globais podem ser eliminados com o uso apropriado de tipos abstratos de dados.

As fases de especificação e de projeto são muito importantes para que se obtenha um produto final confiável, correto e de fácil manutenção. Para apoio ao projetista de sistemas nessas duas fases têm sido utilizados vários métodos, bem como abordagens gerais (descendente, ascendente, etc.) e heuristicas especificas (tamanho do módulo, número de subordinados, etc.).

Um desses métodos, de autoria de Michael Jackson e outros, denominado Jackson Systems Development (JSD) [Ja83,Ca86], difere dos demais por sua base teórica mais sólida e também por sua originalidade. Esse método tem recebido bastante atenção na literatura especializada, tanto através de publicações de seus próprios autores, como de outros pesquisadores [Pe85,Za84,].

Wiener e Sincovec [Wi84] classificam os métodos para análise e projeto de software em dois tipos: pré e pós linguagens do tipo ADA e MODULA-2. No primeiro tipo estão incluídos métodos mais antigos, como Análise e Projeto Estruturado, HIPO, JSP e também JSD. O segundo tipo, segundo eles, é composto por métodos voltados para projetos modulares e orientados para objetos. A fundamentação para essa classificação é o fato de que muitas das atividades e decisões que antes faziam parte só do método são agora parte integrante dessas linguagens, através de seu apoio à abstração de dados.

2. JSD COMO APOIO À PROGRAMAÇÃO MODULAR E ORIENTADA A OBJETOS

A classificação de JSD no grupo dos métodos "antes" de Módula-2 e ADA motivou o desenvolvimento de um experimento, implementando em Modula-2 [Th85] um sistema especificado em JSD, com o objetivo de verificar de que forma ambos poderiam ser utilizados em conjunto e também para validar a classificação de Wiener e Sincovec. Para isso, um sistema de biblioteca, apresentado parcialmente em [Ca86], foi escolhido para ser implementado, após ter tido completada sua especificação. A implementação do sistema está relatada no documento [Sa88a] e a descrição geral do experimento, bem como suas conclusões, encontram-se em [Sa88b].

JSD tem dois conceitos principais : ENTIDADES e AÇÕES. A modelagem em JSD tem inicio com a definição das entidades relevantes para o sistema e depois as ações executadas ou sofridas pelas entidades. Num sistema de biblioteca, duas entidades importantes são LEITOR e LIVRO, por exemplo, e algumas das ações de leitor seriam: inscrever-se, emprestar, renovar, devolver, etc., algumas delas comuns à entidade LIVRO. As ações devem ser ordenadas cronológicamente e representadas por diagramas hierárquicos.

Nas fases seguintes do método, o sistema deve ser especificado na forma de uma rede de processos sequenciais que se comunicam através de troca de mensagens e inspeção ao estado de cada processo. A fase final corresponde à transformação da especificação numa hierarquia de procedimentos implementáveis em um ou mais processadores.

Pode-se perceber que o conceito de ENTIDADE corresponde a um certo tipo de dado e as AÇÕES correspondem a operações que são efetuadas sobre esse tipo de dado. Dessa forma, a idéia básica da implementação efetuada foi criar um TAD para cada entidade, mapeando o projeto de implementação em JSD para as estruturas de módulos de Módula-2.

As regras principais criadas para mapear JSD em uma implementação em Modula-2 são as seguintes:

1. O articulador do sistema (scheduler) deve ser mapeado para um módulo (MODULE), o qual importa os tipos de dados e os procedimentos associados, que estão isolados em outros módulos.

2. Cada entidade definida por JSD deve ser codificada como um tipo de dado: uma estrutura de arquivo contendo uma estrutura de registro (o vetor de estado do processo que modela a entidade). Cada ação realizada pela entidade

deve ser mapeada para uma operação sobre o tipo de dados no qual a entidade foi mapeada. Tanto o tipo de dados quanto os procedimentos devem ser encapsulados num par de módulos do tipo DEFINITION/IMPLEMENTATION, compiláveis independentemente.

3. Todas as funções do sistema, exceto as embutidas, podem ser acondicionadas num módulo independente, do tipo DEFINITION/IMPLEMENTATION, com o nome geral de "funções". (Outra alternativa implementar cada função como operação associada à entidade relacionada à função, mas como há funções que inspecionam o vetor de estado de mais de uma entidade, essa solução não foi adotada, mas pode vir a ser considerada em certas circunstâncias).

4. Os processos de modelagem, como LIVRO-1 e LEITOR-1, invocam os procedimentos relativos às ações nos lugares apropriados de seu texto estruturado. Os procedimentos de ações também podem ser invocados diretamente de outros procedimentos, como é o caso de funções interativas.

5. As rotinas auxiliares, tais como as que implementam o diálogo com o usuário, as que executam consistências, etc., devem ser isoladas num módulo à parte, também do tipo DEFINITION/IMPLEMENTATION. Isso dá maior legibilidade aos procedimentos, fazendo com que sejam implementados de forma bastante próxima ao texto estruturado da especificação.

O diagrama mostrado na figura 1 é adaptado de [Ca86] e ilustra o Diagrama de Implementação do Sistema de Biblioteca, de acordo com as diretrizes de JSD. O diagrama da figura 2 mostra o Projeto Modular do Sistema (Modular Design Chart) para implementação do Sistema de Biblioteca. Este diagrama é sugerido por Wiener e Sincovec em seu livro, para representar projetos modulares de sistemas, orientado a objetos. Essa é a única ferramenta sugerida em [Wi84] como apoio a esse tipo de projeto.

O diagrama da figura 2 é o resultado obtido após a aplicação das regras 1 a 5 sugeridas acima, ao sistema de biblioteca especificado em JSD. Esse diagrama representa a organização modular do sistema, com um módulo principal e os demais módulos estruturados em torno de um tipo abstrato de dados. O diagrama mostra também os procedimentos que cada módulo usa (representados pelos pontos pretos nas linhas de conexão).

3. CONCLUSÕES

O experimento mostrou que JSD pode ser usado como um método orientado a objetos. Deve-se notar que esse termo, como usado por Wiener e Sincovec, refere-se mais à programação com tipos abstratos de dados, sem considerar o conceito de "herança", inexistente em ADA e Módula-2. Para estudar o comportamento de

JSD com linguagens do tipo Smalltalk, por exemplo, outros experimentos estão sendo planejados.

O experimento mostrou também que é bastante direta a criação de objetos (estruturas de dados + procedimentos associados) em Modula-2 que correspondem às abstrações utilizadas em JSD.

A autora tem a firme convicção de que um projetista, mesmo usando linguagens como Modula-2 e ADA, necessita de muito mais ferramentas para apoio ao projeto de software do que apenas o Diagrama de Projeto Modular do sistema, sugerido em [Wi84]. Em especial, o projetista deve basear-se em algum modelo mental de raciocínio para escolher um conjunto apropriado de tipos de dados e de procedimentos para resolver um dado problema. JSD pode oferecer o modelo conceitual para apoio a essa tarefa, de tal forma que ela possa ser conduzida de forma sistemática.

O conjunto de diagramas e textos estruturados sugeridos por JSD é conciso, mas de manutenção trabalhosa. Várias vezes no decorrer do projeto, diagramas e/ou textos precisaram ser modificados e quase sempre descobriu-se mais tarde que algum detalhe foi esquecido, como por exemplo: acrescenta-se um novo fluxo de dados e esquece-se de inserir uma sentença do tipo "Escrever ‹fluxo de dados›" no texto estruturado do processo que gera o fluxo de dados. Com o auxílio de alguma ferramenta automática para apoiar o uso do método, esses problemas podem diminuir.

O critério de modularização induzido por JSD faz com que fique claramente especificado quais são os processos (procedimentos) que têm o direito de modificar os dados relativos a uma entidade, facilitando a depuração do sistema e possíveis manutenções no futuro. Os dados armazenados por pura necessidade de processos funcionais também ficam explicitados.

Modula-2 também facilitou bastante a implementação, principalmente os testes e o desenvolvimento da programação: depois de especificar em JSD o sistema alvo, sabem-se os tipos de dados e os procedimentos e, consequentemente, os módulos de definição podem ser criados; a implementação das ações e das funções pode ser feita incrementalmente sem necessidade de mudar os modulos de definição.

AGRADECIMENTO: Este trabalho foi realizado dentro de um programa de Iniciação científica, com apoio da FAPESP, sob orientação do Prof. Paulo C. Masiero.

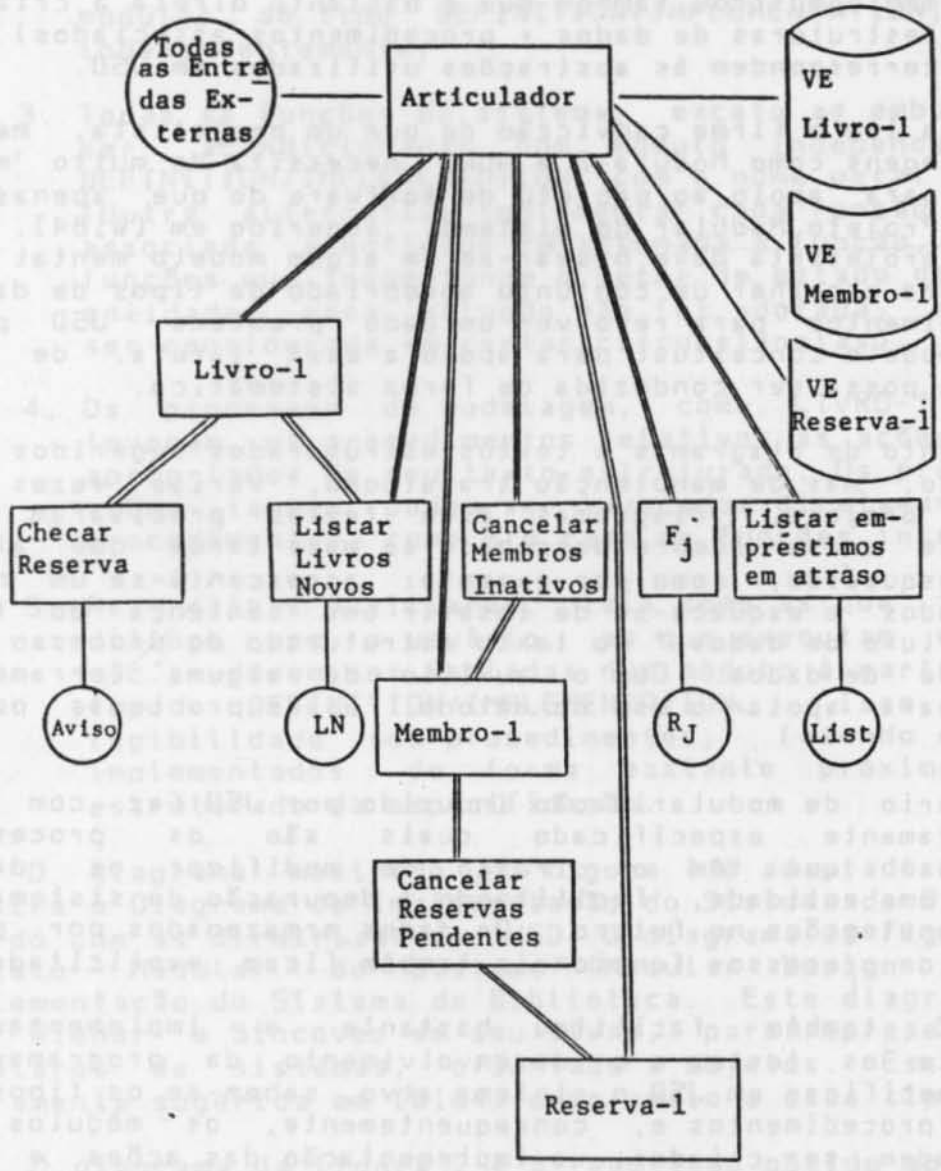

Fig. 1 - Diagrama de Implementação do Sistema

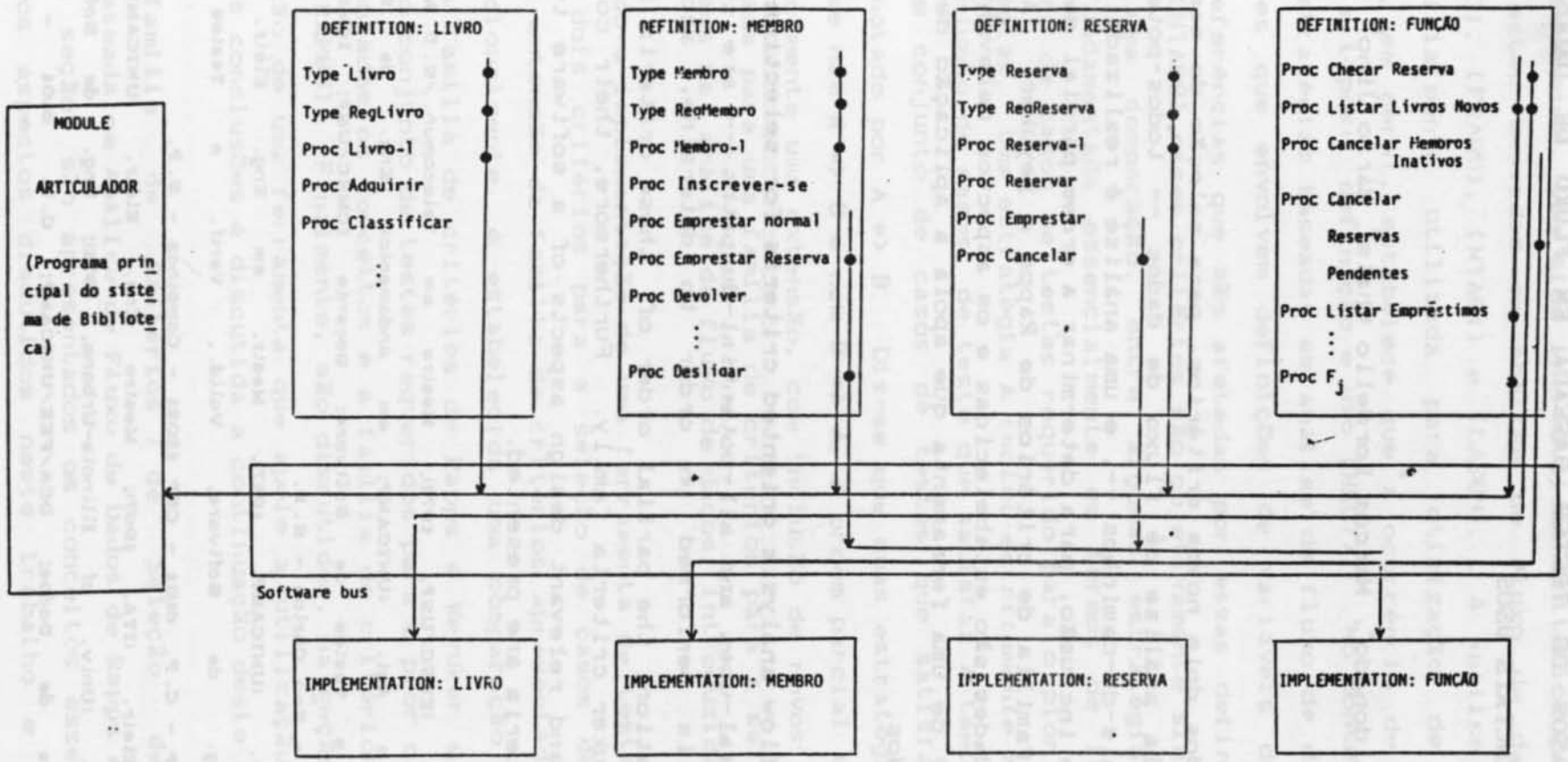

Figura 2 - Diagrama de Estrutura Modular