XSBIAGRO 21 a 23 de outubro de 2015
X Congresso Brasileiro de Agroinformática

Especificação e instanciação de modelos matemáticos hierárquicos utilizando XML no framework de simulação MaCSim

Adauto Luiz Mancini[1], Luis Gustavo Barioni[1], Jair Bortolucci Junior[2], Luiz Victor Stefani Tinini[3], Jheniffer Jordão Leonardi[4], Jacomo Giovanetti Minto Neto[4]

[1] Laboratório de Matemática Computacional, Embrapa Informática Agropecuária, Campinas, São Paulo, Brasil, adauto.mancini@embrapa.br, luis.barioni@embrapa.br

[2] Universidade Estadual de Campinas, Campinas, São Paulo, Brasil, jairbortolucci@hotmail.com

[3] Pontifícia Universidade Católica de Campinas, Campinas, São Paulo, Brasil, lv_luiz.victor@hotmail.com

[4] Faculdade de Tecnologia de Americana, Americana, São Paulo, Brasil, jhenifferleonardi@gmail.com, jacomogiovanettimn@gmail.com

RESUMO

Existe, atualmente, uma tendência de modularização do desenvolvimento de modelos matemáticos e seu posterior acoplamento, particularmente em sistemas complexos. O *design*, documentação e compartilhamento de modelos matemáticos para simulação de sistemas, entre os membros de uma comunidade científica ou equipe de trabalho, é facilitado por arquivos não executáveis de especificação dos modelos hierárquicos. Esta solução também facilita o uso integrado de ferramentas computacionais para a manipulação dos modelos, como interfaces para visualização e edição das especificações, construção de simuladores a partir das especificações, repositórios de modelos e aplicativos clientes com uso dos modelos ou simuladores. A equipe do Laboratório de Matemática Computacional da Embrapa Informática Agropecuária optou pela metalinguagem *Extensible Markup Language* (XML), para desenvolver um padrão de especificação de modelos atômicos e hierárquicos para o *framework* de simulação MaCSim. Para o uso do padrão foram desenvolvidas no *framework* classes para leitura/escrita das especificações dos modelos em XML e a instanciação em tempo de execução de objetos correspondentes aos modelos permitindo a construção de simuladores dos modelos especificados em tempo de execução. Esta solução mostrou-se adequada, oferecendo maior

produtividade e flexibilidade no desenvolvimento de modelos hierárquicos no *framework* MaCSim.
PALAVRAS-CHAVE: Representação de Sistemas Dinâmicos, Metalinguagem, Reuso.

ABSTRACT

There is now a trend of modularization of the development of mathematical models and their subsequent engagement, particularly in complex systems. The design, documentation and sharing of mathematical models for simulation systems, among members of a scientific community or work team, is facilitated by non-executable file specification of hierarchical models. This solution also facilitates the integrated use of computational tools for the manipulation of models, such as interfaces for viewing and editing of specifications, building simulators from the specifications, models, repositories and applications customers with use of models or simulators. The staff of the Laboratory of Computational Mathematics of Embrapa Informática Agropecuária opted metalanguage Extensible Markup Language (XML), to develop a standard specification of atomic and hierarchical models for MaCSim simulation *framework*. For standard use were developed in the *framework* classes to read / write the specifications of XML models and instantiation at runtime of objects corresponding to models allowing the construction of simulators of the specified models at runtime. This solution was adequate, offering increased productivity and flexibility in the development of hierarchical models in MaCSim *framework*.
KEYWORDS: Dynamic System Representation, Metalanguage, Reuse.

INTRODUÇÃO

Sistemas de produção agropecuária envolvem uma série de processos biofísicos e econômicos em diversos níveis de agregação ou granularidade. Modelos são, portanto, ferramentas essenciais para compreender e prever o comportamento desses sistemas sob condições de interesse (SAMIK et al., 2011).

Modelos podem ser representados no nível conceitual, por textos e diagramas. Entretanto, modelos podem ser mais rigorosamente representados por fórmulas matemáticas, facilitando a simulação computacional (ZIEGLER et al., 2000). Diferencia-se o modelo como especificação de um sistema e o simulador como o programa computacional (i.e. conjunto de algoritmos) que pode interpretar a especificação do modelo e gerar resultados numéricos a partir dele. Esse conceito implica na possibilidade de armazenamento de modelos matemáticos em arquivos não executáveis. Embora a representação matemática seja suficiente para a simulação

computacional, conceitos de engenharia de software, incluindo modularidade, baixo acoplamento e reuso são de grande importância na implementação de simuladores de maior porte. Além disso, a representação de sistemas de maneira hierárquica permite uma clara aderência à teoria de sistemas e facilita a compreensão de processos em diferentes níveis de agregação (ZIEGLER et al., 2000). Nesse contexto, o advento da programação orientada a objetos foi extremamente importante no processo de organização e estruturação da implementação de simuladores.

Frameworks de simulação, em português arcabouços de simulação, como o MaCSim (MANCINI et al., 2014) têm por finalidade prover as funcionalidades necessárias para o desenvolvimento de simuladores e são responsáveis por definir o fluxo de controle da simulação.

A representação de sistemas, pelo uso de modelos matemáticos em simuladores orientados a objetos, vai além da representação matemática das equações. É preciso definir também o conjunto de modelos componentes que compõem o sistema, o modo como eles são acoplados e dispostos hierarquicamente, bem como a ordem de execução.

Este trabalho descreve as novas funcionalidades do *framework* MaCSim, desenvolvidas para a construção de simuladores em tempo de execução e a adequação do *framework* para o uso de *Extensible Markup Language* (XML), uma metalinguagem de marcação (PITTS-MOULTIS, 2000) amplamente aceita e suportada para o armazenamento e recuperação da especificação de modelos hierárquicos.

MATERIAL E MÉTODOS

O MaCSim é um *framework* de simulação orientado a objetos, implementado com a linguagem de programação C++, utilizado neste trabalho. O MaCSim permite o desenvolvimento de modelos acoplados (compostos) pela instanciação de modelos atômicos (componentes) e pelo estabelecimento de conexões e hierarquia entre eles. As conexões são estabelecidas por meio de ponteiros entre portas de entrada e saída de dados dos modelos. A hierarquia é possibilitada por um auto relacionamento de composição na classe "*Model*", que é a classe base de todos os modelos implementados no *framework*. Os componentes de um modelo podem ser instanciados via código do método construtor. Esse procedimento requer, entretanto, que o modelo seja uma classe especializada a partir da classe "*Model*". É possível gerar o simulador construído com o MaCSim como uma *Dynamic-link library* (DLL) para a integração com outras ferramentas ou linguagens para realizar simulações, assim como o armazenamento e a recuperação de modelos.

A biblioteca compilada pode ser usada por diferentes aplicativos clientes, por exemplo um programa escrito na linguagem de programação R ou um aplicativo com interface gráfica. Com os objetivos de estender o *framework* para: a) facilitar a construção de modelos acoplados sem requerer muito conhecimento de programação em C++; b) permitir a construção de modelos em tempo de execução; c) criar simuladores a partir da leitura da especificação do modelo ou escrever a especificação a partir de um simulador existente; projetou-se e implementou-se as novas classes *Factory* e *Assembler* para os objetivos a), b) e os métodos *WriteModelToXml* e *ReadModelFromXml* da classe *Interface* para o objetivo c).

A classe *Factory*, ou "Fábrica", instancia modelos atômicos em tempo de compilação, a partir do código-fonte desses modelos no MaCSim devido à ausência de um interpretador de equações.

A classe *Assembler*, ou "Montadora", foi implementada para a construção de modelos acoplados a partir de modelos atômicos. É responsável por conectar os modelos atômicos e definir a disposição hierárquica deles, além de instanciar em tempo de execução estes modelos e suas respectivas portas de entrada e saída construídos previamente pela classe *Factory*. Cada container é uma instância da classe *Model* que suporta especificamente modelos de sistemas dinâmicos, incluindo hierarquia e propagação de mensagens para realização de cálculos e resposta a eventos. A classe possui comandos simples que fazem referência à estrutura hierárquica do modelo acoplado, não exigindo conhecimento profundo da programação do *framework* para seu uso.

Os principais métodos para construção de modelos hierárquicos da classe *Assembler* são:

- *CreateContainer(Nome Do Novo Recipiente, Nome Do Recipiente Pai)* - cria um modelo hierárquico com identificação *Nome Do Novo Recipiente*, que será adicionado como um modelo atômico de *Nome Do Recipiente Pai*. Se o modelo *Nome Do Novo Recipiente* for a raiz da hierarquia, atribui-se "*NULL*" para o nome do pai. O modelo acoplado *Nome Do Novo Recipiente* é criado vazio, adicionando-se posteriormente os modelos atômicos que podem ser criados antes ou depois do modelo acoplado. Uma vez criados os objetos componentes e o *container*, pode-se chamar os métodos de adição de modelos. Um modelo pode ser simultaneamente componente e *container* de outros modelos, criando-se tantos níveis hierárquicos quantos forem necessários.

- *AddSubModel(Nome Do Recipiente Pai, Nome Do Recipiente Filho)* – adiciona o modelo *Nome Do Recipiente Filho* na lista de componentes do modelo *Nome Do Recipiente Pai.*
- *AddAtomicSubModel(Nome Do Recipiente Pai, Nome Da Classe Do Filho, Nome Do Filho)* – cria um modelo atômico, instância da classe *Nome Da Classe Do Filho* com identificação *Nome Do Filho* e insere como componente de *Nome Do Recipiente Pai.* Um modelo atômico é aquele que contém uma ou mais variáveis de estado do sistema em estudo, calculadas a partir das equações diferenciais do modelo matemático. Na versão corrente do *framework*, um modelo atômico é criado por uma chamada do objeto *Assembler* para o objeto *Factory*. O objeto *Factory* recebe o *Nome Da Classe Do Filho*, e instancia um objeto da classe adequada, que será adicionado como modelo componente de *Nome Do Recipiente Pai* no método *AddAtomicSubModel* do objeto *Assembler.*
- *CreateAutomaticConnections(Nome Do Recipiente)* - cria no objeto com identificação *Nome Do Recipiente* portas de entrada e de saída, com nomes correspondentes as portas dos objetos dos modelos atômicos e conecta as portas de entrada de *Nome Do Recipiente* com as portas de entrada dos modelos atômicos que tenham a mesma identificação (título). Faz o procedimento equivalente para as portas de saída.
- *ConnectPort(Nome Do Recipiente Pai, Nome Do Primeiro Recipiente Filho, Nome Da Porta Saída Do Primeiro Filho, Nome Do Segundo Recipiente Filho, Nome Da Porta Entrada Do Segundo Filho)* – conecta à porta de saída *Nome Da Porta Saída Do Primeiro Filho* do componente *Nome Do Primeiro Recipiente Filho* com a porta de entrada *Nome Da Porta Entrada Do Segundo Filho* do componente *Nome Do Segundo Recipiente Filho,* pertencendo sendo ambos os componentes *Nome Do Recipiente Pai.*

Para o armazenamento e a recuperação a partir de modelos especificados em arquivo, foram desenvolvidos no *framework* métodos para escrita e leitura em tempo de execução com o auxílio da biblioteca *CMarkup 11.5* para manipulação de arquivos XML (http://www.firstobject.com/dn_markup.htm).

Os principais métodos adicionados à classe *Interface* para a representação de modelos em XML são:

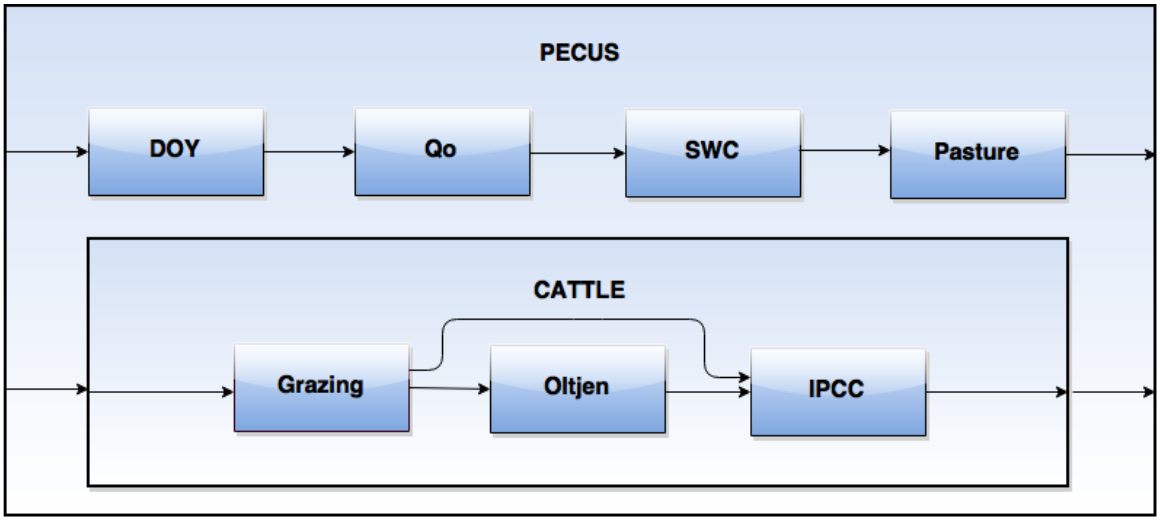
- *WriteModelToXml* – este método percorre a hierarquia de um modelo acoplado, a partir de um ponteiro para um objeto em memória da classe *Model*, transcrevendo a especificação para um arquivo XML. As *tags* principais do modelo são: "*Model*", "*Ports*", "*SubComponents*" e "*Connections*". O método é chamado recursivamente para os componentes do modelo para a composição da *tag* "*SubComponents*".
- *ReadModelFromXml* – lê um modelo descrito no arquivo XML e faz chamada aos métodos da classe *Assembler* para compor o modelo acoplado no *framework*.

Conforme acima mencionado, a classe *Assembler* permite a composição e a instanciação de modelos em tempo de execução. Quando a biblioteca é carregada pela aplicação cliente, é criada na biblioteca os *singletons Assembler* e *Factory*. A aplicação cliente pode então disparar procedimentos da biblioteca que desencadeiam a chamada de métodos no MaCSim para a criação da estrutura desejada para a construção (instanciação) do modelo acoplado desejado, por exemplo lendo uma especificação XML.

RESULTADOS E DISCUSSÃO

Como um exemplo do uso dos comandos da classe *Assembler*, é mostrada no Quadro 1, a construção do modelo acoplado PECUS que simula um sistema de produção pecuária e sua arquitetura, representada na forma de diagrama de blocos, na Figura 1. A transcrição do modelo PECUS (Figura 1) em XML é mostrada no Quadro 2.

Figura 1 - Diagrama de modelos hierárquicos gerado pelo código do quadro 1. PECUS é o modelo principal, DOY, Qo, SWC, Pasture e CATTLE, são modelos atômicos do modelo PECUS. Grazing, Oltjen e IPCC são modelos atômicos do modelo CATTLE.



Fonte: Diagrama desenvolvido pela equipe de programação do projeto PECUS Modelagem Biofísica

No diagrama de blocos da Figura 1, é ilustrado o acoplamento dos modelos componentes realizado pela classe *Assembler* e as conexões entre cada modelo. As conexões podem ser entre o modelo container e seus componentes ou entre os modelos componentes do mesmo container. Essas conexões possuem um fluxo de entrada (*input*) e saída (*output*) de dados.

Quadro 1 – Exemplo de script para construção de um modelo acoplado

```
CreateContainer("PECUS", "NULL");
AddAtomicSubModel("PECUS", "DOY", "DOY");
AddAtomicSubModel("PECUS", "Qo", "Qo");
AddAtomicSubModel("PECUS", "SWC", "SWC");
AddAtomicSubModel("PECUS", "Pasture", "Pasture");
CreateContainer("CATTLE", "PECUS");
AddAtomicSubModel("CATTLE", "Grazing", "Grazing");
AddAtomicSubModel("CATTLE", "Oltjen", "Oltjen");
AddAtomicSubModel("CATTLE", "IPCC", "IPCC");
ConnectPort("CATTLE", "Grazing", "DMI", "Oltjen", "DMI");
ConnectPort("CATTLE", "Grazing", "Ingesta_MEC", "Oltjen", "MEC");
ConnectPort("CATTLE", "Grazing", "Ingesta_Nem", "Oltjen", "NEm");
ConnectPort("CATTLE", "Grazing", "Ingesta_Neg", "Oltjen", "Neg");
ConnectPort("CATTLE", "Oltjen", "IMEnorm", "Grazing", "MEIDemand");
ConnectPort("CATTLE", "Oltjen", "EBW", "IPCC", "PesoVivo");
ConnectPort("CATTLE", "Grazing", "DMI", "IPCC", "IMS");
CreateAutomaticConnections("CATTLE");
AddSubModel("PECUS", "CATTLE");
ConnectPort("PECUS", "DOY", "DOY", "Qo", "DOY");
ConnectPort("PECUS", "Qo", "Qo", "SWC", "Qo");
ConnectPort("PECUS", "SWC", "ISNA", "Pasture", "ISNA");
ConnectPort("PECUS", "Pasture", "LeafMass", "CATTLE", "LeafMass");
ConnectPort("PECUS", "Pasture", "StemMass", "CATTLE", "StemMass");
ConnectPort("PECUS", "Pasture", "DeadMass", "CATTLE", "DeadMass");
ConnectPort("PECUS", "CATTLE", "Leaf_DMI", "Pasture", "Leaf_DMI");
ConnectPort("PECUS", "CATTLE", "Stem_DMI", "Pasture", "Stem_DMI");
ConnectPort("PECUS", "CATTLE", "Dead_DMI", "Pasture", "Dead_DMI");
CreateAutomaticConnections("PECUS");
```

Fonte: Código desenvolvido pela equipe de programação do projeto PECUS Modelagem Biofísica

Quadro 2 – Exemplo de XML construído a partir dos objetos do modelo em execução no *framework* MaCSim.

```
<Model>
        <ClassName>Cls_DynModel</ClassName>
        <ModelName>PECUS</ModelName>
<Ports>
        <Type>Input</Type>
        <Identity>0</Identity>
        <Name>Init_DOY</Name>
        <Value>1</Value>
        …
</Ports>
<SubComponents>
        <Model>
                <ClassName>Cls_DOY_Model</ClassName>
                <ModelName>DOY</ModelName>
         <Ports>
                <Type>Input</Type>
                <Identity>0</Identity>
                <Name>Init_DOY</Name>
                <Value>1</Value>
                …
        </Ports>
        …
        </Model>
</SubComponents>
<Connections>
        <Connection>
                <Identity>0</Identity>
                <ConnectionsOf>PECUS</ConnectionsOf>
                <ModelFrom>DOY</ModelFrom>
                <PortFrom>DOY</PortFrom>
                <ModelTo>Qo</ModelTo>
                <PortTo>DOY</PortTo>
        </Connection>
        …
</Connections>
</Model>
```

Fonte: Código desenvolvido pela equipe de programação do projeto PECUS Modelagem Biofísica

CONCLUSÕES

A especificação dos modelos matemáticos em XML e o desenvolvimento da classe *Assembler* no *framework* de simulação MaCSim permite criar modelos acoplados em tempo de execução.

Isto evita re-compilação, pois antes era preciso a especialização de novas classes para a criação de modelos acoplados. O processo de desenvolvimento é acelerado pelo reuso a partir da especificação textual de modelos, ao invés da necessidade de conhecimento de programação para reuso de código-fonte. Apesar do formato XML ser desenvolvido para uso em máquinas, também pode ser lido por humanos, assim a especificação de modelos acoplados fica mais concisa no XML, não exigindo os conhecimentos específicos da linguagem C++, que seriam necessários no caso da especialização de classes. O fato da especificação no formato XML ser mais conciso permite a criação de um repositório de modelos mais compreensíveis para os usuários.

A instanciação de simuladores correspondentes aos modelos especificados em XML permite o desenvolvimento de outros aplicativos, por exemplo uma interface para visualização e especificação de modelos, bem como o teste e execução dos simuladores construídos. O amplo suporte das diversas linguagens de programação e aplicativos ao XML permite a fácil portabilidade dos modelos do *framework* de simulação para uma vasta gama de ferramentas.

A especificação de modelos pode ser transferida através de um único arquivo XML ao invés de portar todo o código-fonte necessário para a criação dos executáveis dos modelos acoplados, contendo possivelmente muitos arquivos, garantindo assim integridade e praticidade.

É possível criar um repositório de especificações XML para compartilhamento de modelos entre os diferentes usuários, desde que os modelos atômicos contidos nos modelos acoplados se encontrem na versão corrente do executável do *framework* sendo utilizado. A classe *Assembler* permite especificar apenas modelos acoplados, enquanto que modelos atômicos precisam estar compilados no executável do *framework* para que a classe *Factory* possa gerar instâncias das classes dos modelos atômicos.

A interpretação ou compilação de fórmulas matemáticas contidas nos modelos atômicos ainda não é feita diretamente a partir da especificação XML dos modelos, portanto é considerado o desenvolvimento desta funcionalidade futuramente, para que também possam ser criados objetos dos modelos atômicos em tempo de execução.

AGRADECIMENTOS

Agradecemos aos membros da equipe do Laboratório de Matemática Computacional que empenharam esforços, tornando esse artigo possível. À Embrapa Informática Agropecuária pela oportunidade concedida e a todos os outros colaboradores envolvidos.

REFERÊNCIAS

ZEIGLER, B. P.; PRAEHOFER, H.; KIM, T. G.2nd ed. Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. San Diego: Academic Press, 2000. 510 p. ill.

SAMIK, G.; YUKIKO, M.; YOSHIYUKI A.; KUN-YI, H.; HIROAKI, K. Software for systems biology: from tools to integrated platforms, vol. 12. Published Online: Macmillan Publishers Limited, 2011. 821-832 p.

MANCINI, A. L. et al. Arcabouço para desenvolvimento de simuladores de sistemas dinâmicos contínuos e hierárquicos. In: Informação Tecnológica em Agricultura, 2013, Campinas. 19 p. (Embrapa Informática Agropecuária. Boletim de pesquisa e desenvolvimento, 34). Disponível em: <http://www.infoteca.cnptia.embrapa.br/bitstream/doc/981039/1/BolPesq34cnptia.pdf/>. Acesso em: 20 mar. 2015.

PITTS-MOULTIS, N. XML Black Book, 1. ed. São Paulo: MAKRON Books, p. 627, 2000.

HAROLD, E. R; MEANS, S. W. XML in a nutshell, 1. ed. USA: O'Reilly & Associates, Inc, p. 480, 2001.