Boletim de Pesquisa e Desenvolvimento 34
ISSN 1677-9266
dezembro, 2013

Arcabouço para Desenvolvimento de Simuladores de Sistemas Dinâmicos Contínuos e Hierárquicos

Embrapa

ISSN 1677-9266
Dezembro, 2013

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

Boletim de Pesquisa e Desenvolvimento 34

Arcabouço para Desenvolvimento de Simuladores de Sistemas Dinâmicos Contínuos e Hierárquicos

Adauto Luiz Mancini
Luís Gustavo Barioni
Edgard Henrique dos Santos
Fernando Rodrigues Teixeira Dias
Jefferson William dos Santos
Luiz Lino Bertanha de Abreu
Luiz Victor Stefani Tinini

Embrapa Informática Agropecuária
Campinas, SP
2013

Embrapa Informática Agropecuária
Av. André Tosello, 209 - Barão Geraldo
Caixa Postal 6041 - 13083-886 - Campinas, SP
Fone: (19) 3211-5700 - Fax: (19) 3211-5754
www.cnptia.embrapa.br
cnptia.sac@embrapa.br

1ª edição
on-line 2013

Dados Internacionais de Catalogação na Publicação (CIP)
Embrapa Informática Agropecuária

Arcabouço para desenvolvimento de simuladores de sistemas dinâmicos contínuos e hierárquicos / Adauto Luis Mancini... [et al.] .- Campinas : Embrapa Informática Agropecuária, 2013.
19 p. il.: cm. - (Boletim de pesquisa e desenvolvimento / Embrapa Informática Agropecuária, ISSN 1677-9266; 34).

1. Modelagem matemática. 2. Software de simulação. 3. Simulação orientada a objeto. I. Mancini, Adauto Luís. II. Embrapa Informática Agropecuária. III. Título. IV. Série.
003.3 CDD 21 ed.

Sumário

Arcabouço para Desenvolvimento de Simuladores de Sistemas Dinâmicos Contínuos e Hierárquicos

Adauto Luiz Mancini[1]
Luís Gustavo Barioni[2]
Edgard Henrique dos Santos[3]
Fernando Rodrigues Teixeira Dias[4]
Jefferson William dos Santos[5]
Luiz Lino Bertanha de Abreu[6]
Luiz Victor Stefani Tinini[7]

Resumo

Um arcabouço (*framework*) orientado a objetos para o desenvolvimento de simuladores de sistemas dinâmicos contínuos com suporte à hierarquia é descrito. O arcabouço surgiu da necessidade de uma ferramenta para fa-

[1] *Cientista da computação, mestre em Ciências, pesquisador da Embrapa Informática Agropecuária, Campinas, SP, adauto.mancini@embrapa.br*

[2] *Engenheiro agrônomo, doutor em Ciência animal e pastagens, pesquisador da Embrapa Informática Agropecuária, Campinas, SP, luis.barioni@embrapa.br*

[3] *Cientista da computação, analista da Embrapa Informática Agropecuária, Campinas, SP, edgard.santos@embrapa.br*

[4] *Engenheiro eletrônico, mestre em Administração de empresas, pesquisador da Embrapa Pantanal, Corumbá, MS, fernando.dias@embrapa.br*

[5] *Graduando em Análise e desenvolvimento de sistemas, estagiário da Embrapa Informática Agropecuária, Campinas, SP, jefferson.santos16@fatec.sp.gov.br*

[6] *Graduando em Análise e desenvolvimento de sistemas, estagiário da Embrapa Informática Agropecuária, Campinas, SP, lino.bertanha.br@gmail.com*

[7] *Graduando em Sistemas da informação, estagiário da Embrapa Informática Agropecuária, Campinas, SP, lv_luiz.victor@hotmail.com*

cilitar e padronizar o desenvolvimento de modelos de simulação baseados em processo em projetos de pesquisa na Embrapa. Um dos requisitos do desenvolvimento foi modularidade e simplicidade de código para facilitar o desenvolvimento de modelos por equipes multidisciplinares de pesquisa e implementação por grupos de estagiários/bolsistas com alta rotatividade. Também provê desempenho superior a ferramentas de modelagem típicas, que geralmente tem código interpretado. Modelos componentes, geralmente desenvolvidos por equipes de especialistas em processos específicos, podem ser desenvolvidos independentemente e conectados posteriormente, sequencialmente ou agregados em hierarquias. O arcabouço permite compilar simuladores como bibliotecas e provê uma interface geral que permite simulações serem executadas por aplicações clientes, como interfaces gráficas do usuário, bancos de dados, pacotes estatísticos ou matemáticos. Contrastando com outros arcabouços existentes, este não armazena trajetórias de variáveis, permitindo à aplicação cliente obter os valores de saída durante a simulação por meio de *callbacks*. A aplicação cliente armazena as trajetórias de variáveis na forma mais conveniente para seus propósitos específicos. O arcabouço suporta simulação contínua, discreta e híbrida.

Termos para indexação: Simulação orientada a objetos, software de simulação, modelagem matemática, simulação baseada em processo.

A Framework for Development of Continuous Dynamical Systems with Support to Hierarchy

Abstract

The object-oriented simulation framework *(SF) presented herein arose from the need for a tool to facilitate and standardize the development of process-based simulation models in research projects at Embrapa (Brazilian Agricultural Research Corporation). This SF targeted modularity and simplicity of code to facilitate model development by multidisciplinary research teams and implementation by high turnover groups of undergraduate student trainees. It also provides higher performance than typical interpreted modeling tools. Model components, typically developed by teams of experts in specific processes, can be developed independently and later connected, sequentially or aggregated in a hierarchical way. The SF allows compiling simulators as libraries and provides a general interface to allow simulations to be carried out by client applications, e.g. graphical user interfaces, databases, statistical or mathematical packages. In contrast to other existing frameworks, this SF does not store the trajectories of the variables but allows the client application to get the values of the outputs along the simulation through the use of* callbacks*. The client application can store the variables trajectories in the most convenient way for its specific purpose. The SF supports continuous, discrete-event and hybrid simulations.*

***Index terms**: Object-oriented simulation, simulation software, mathematical modelling, process-based simulation*

Introdução

O uso de modelagem e simulação tem sido aplicado crescentemente em ciências, engenharia e educação (BOOTE et al., 1998). A evolução da tecnologia da informação gerando computadores cada vez mais poderosos permitem simular processos com maior nível de detalhamento, resolução temporal e espacial, quantidade de fontes de variação individuais ou estocásticas, e métodos heurísticos. O desenvolvimento de modelos complexos, como o desenvolvimento genérico de software de grande escala, ainda é desafiador e inúmeros paradigmas e ferramentas surgem para atender estes desafios de modo efetivo.

Software para simulação deveria permitir reúso eficiente de modelos em diferentes contextos e escalas (MOORE et al., 2007). Jones et al. (2001) acrescentam a esta opinião que modelos componentes deveriam poder ser adiciononados, modificados e mantidos com esforço mínimo. Entre as ferramentas de modelagem e simulação, existem arcabouços que permitem codificar e compilar modelos pelo uso de linguagens de programação genéricas. Estes têm a vantagem de permitir o uso de toda a capacidade da linguagem e suavizar sua integração com aplicações clientes. Além disto, o desempenho é usualmente melhor comparado com a implementação por ferramentas que geram modelos escritos em linguagens interpretadas. O arcabouço descrito neste trabalho advém da necessidade de uma ferramenta para facilitar e padronizar o desenvolvimento de modelos de simulação baseados em processo em projetos de pesquisa na Empresa Brasileira de Pequisa Agropecuária (Embrapa). O primeiro projeto almejado pelo arcabouço é o projeto Pecus[8], que objetiva avaliar a emissão de gases de efeito estufa pela pecuária. No desenvolvimento de modelos de sistemas para o projeto Pecus, cada processo biofísico é endereçado a um time diferente de pesquisadores. Após equipes de especialistas de domínio terem prototipado, avaliado, calibrado e modificado com sucesso a estrutura dos modelos componentes candidatos, os modelos precisam ser integrados para produzir o modelo completo do sistema. No contexto do projeto Pecus, a integração dos modelos componentes é conduzida por uma equipe composta por estagiários. Este não é

[8] *Disponível em: <http://www.cppse.embrapa.br/redepecus>.*

um trabalho fácil, pois é necessário conectar entradas e saídas de modelos componentes de vários processos modelados independentemente. Enfrentando este desafio, os autores decidiram desenvolver o arcabouço descrito neste trabalho.

Metodologia

Os principais requisitos para o arcabouço foram modularidade, suporte à hierarquia na estruturação de modelos, padronização de código e reuúso de modelos componentes. Como a maioria dos arcabouços correntes de simulação, por exemplo JDEVS (FILIPPI; BISGAMBIGLIA, 2003); VLE (QUESNEL et al., 2009), o paradigma de orientação a objetos foi escolhido para atender estas necessidades. Outros requisitos levantados pela equipe de desenvolvimento foram:

- suporte à simulação contínua, discreta e híbrida.
- especificação textual padronizada dos modelos para facilitar a interação com especialistas de domínio não programadores.
- código de baixa complexidade para facilitar seu entendimento por alunos de graduação.
- separação do código de controle da simulação do código para implementação de modelos.
- interface simples para facilitar a comunicação entre a aplicação cliente e o simulador.
- alto desempenho computacional via código executável ao invés de interpretado.

A concepção do projeto do arcabouço de software permite o desenvolvimento independente do simulador (os algoritmos de controle da simulação e dos modelos) (NUTARO, 2011) e da aplicação cliente (por exemplo, uma interface gráfica específica do usuário ou um aplicativo genérico, como R). Isto é possibilitado compilando um simulador produzido no arcabouço (controle da simulação e o modelo específico do usuário) como uma biblioteca dinâmica que pode ser chamada pela aplicação cliente. Isto é útil porque: a) um

mesmo simulador pode ser usado por diferentes aplicações; b) o simulador e a aplicação cliente podem ser desenvolvidos por times diferentes usando possivelmente diferentes linguagens de programação; c) o simulador pode ser usado por desenvolvedores de software sem necessidade de conhecimento profundo sobre o código do simulador.

O mesmo simulador pode ser usado para diferentes propósitos, atendendo aplicações clientes distintas (executadas independentemente) e dados da simulação podem ser armazenados de diferentes modos. Por exemplo, um simulador do sistema de uma fazenda pode ser usado para uma simulação determinística em um sistema de suporte à decisão para uma localidade única mas também para uma simulação em grade para uma área geográfica mais abrangente e cenários de mudança climática de longo prazo. Estimação de parâmetros, avaliação de modelo, simulações estocásticas, também são beneficiadas por se ter o simulador compilado como uma biblioteca e usado por uma aplicação cliente.

Além do simulador propriamente, a aplicação cliente precisa implementar ou requerer outros componentes de software para apropriadamente fornecer ou solicitar dados para/do simulador e armazená-los ou apresentá-los (interface gráfica do usuário, banco de dados, rede de comunicação). A separação do simulador dos outros componentes de software pode resultar em um esforço mais eficiente e especializado conduzido paralelamente por equipes dedicadas. Neste contexto, componentes de software, que não sejam o simulador propriamente, podem ser desenvolvidos por instituições parceiras.

Um protocolo de comunicação padrão, definido pelo arcabouço, facilita o desenvolvimento e minimiza erros, uma vez que chamadas entre os componentes da aplicação cliente e o simulador não precisam ser redesenhadas para cada uso específico das aplicações clientes. A produtividade também pode ser aumentada, porque cada aplicação poderá usar o mesmo protocolo com o simulador, de modo que desenvolvedores precisarão aprendê-lo apenas uma vez e não será necessário conhecimento detalhado sobre o código do simulador.

O arcabouço foi concebido para que a aplicação cliente se encarregue do armazenamento de dados. Esta abordagem difere daquela descrita por

Bolte (1998), em que o armazenamento de dados é feito pelo software simulador. O desacoplamento dos componentes simuladores do serviço de armazenamento de dados provê flexibilidade, pois a aplicação cliente pode armazenar as saídas do modo mais adequado para atender sua demanda de análise (memória, arquivo texto ou banco de dados). Também melhora a eficiência porque evita armazenamento desnecessário dos dados de cada iteração gravando-os em disco a cada instante, por exemplo.

Dois pacotes principais compõem o software: Controle de Simulação e Objetos de Simulação, além da classe SF_Interface (Figura 1). O Controle de Simulação fornece a infraestrutura de gerenciamento do processo de simulação (incluindo as classes *Control*, *Event and Event_Manager*).

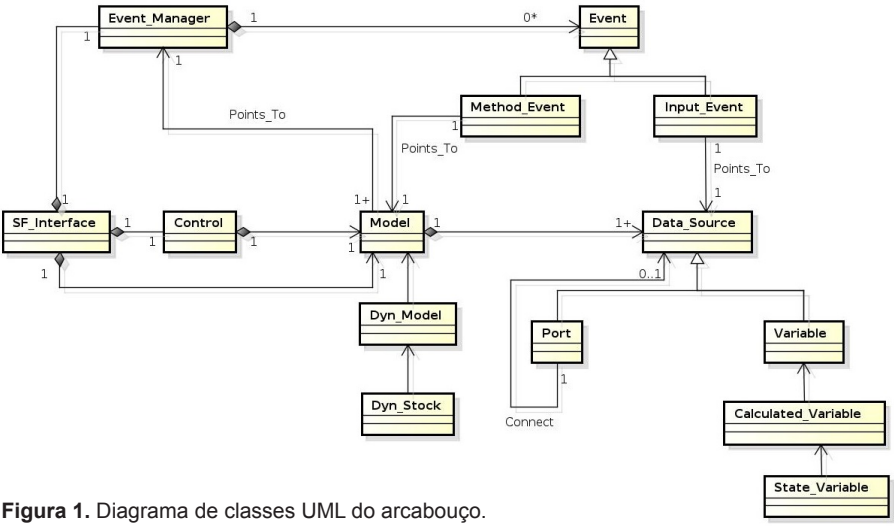

Figura 1. Diagrama de classes UML do arcabouço.

Objetos de Simulação provê as classes bases (*Model*, *Data_Source*, *Port* e *Variable*) para implementar o modelo do sistema de interesse do usuário. A classe SF_Interface é responsável pela construção e destruição do modelo principal e também por instanciar o Controle de Simulação. Ela também gerencia toda interação entre a aplicação cliente e o simulador (por exemplo: atribui os valores iniciais, eventos, passo de tempo e duração de uma simulação – Figura 2).

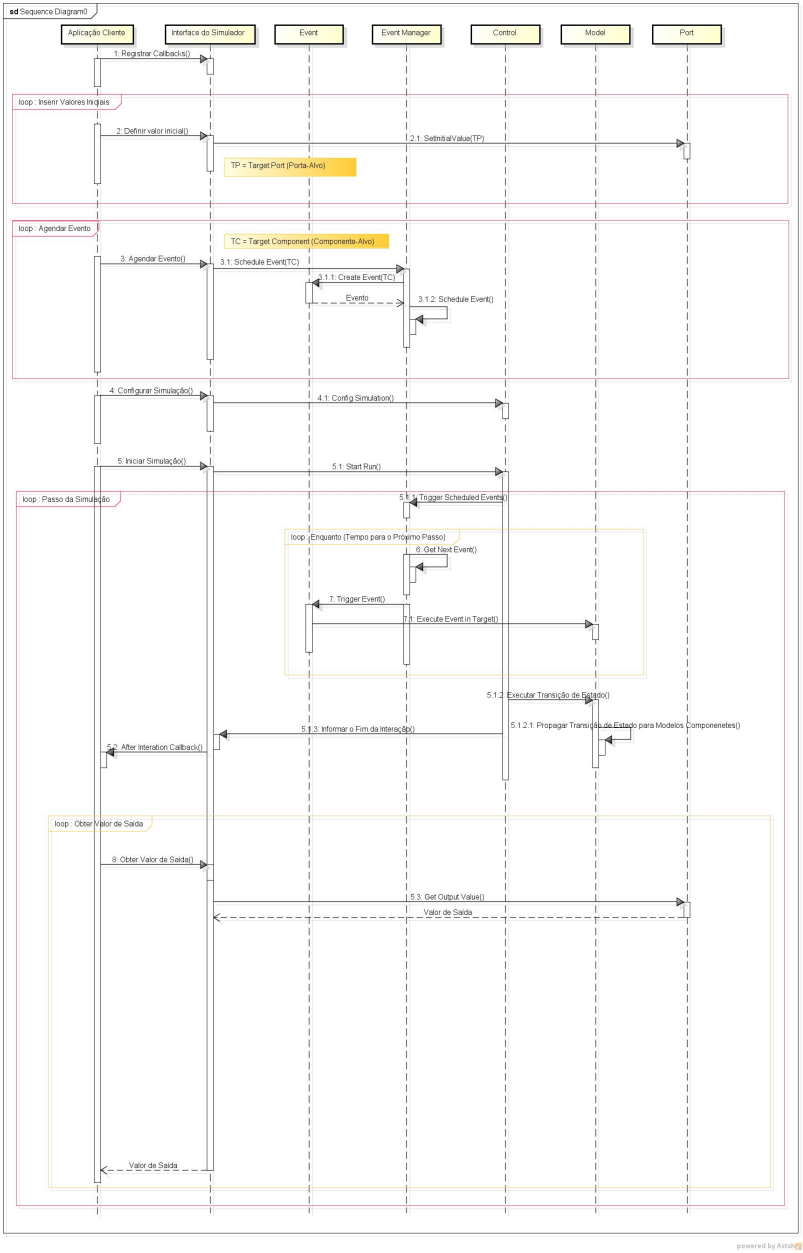

Figura 2. Diagrama de sequência UML de execução de um simulador.

A classe *Control* contém o relógio do simulador e requisita ao *Event_ Manager* para disparar eventos que transitam o estado do *Model* e também dispara a chamada das *callbacks* relacionadas ao progresso da simulação (eventos *before_iterate* e *after_iterate*, por exemplo). A classe *Event*, base das subclasses *Input_Event* e *Method_Event*, contém atributos para o tempo de ocorrência do evento, um ponteiro para o objeto alvo e um literal contendo o nome do método do objeto alvo a ser chamado. Quando um evento é criado pelo *Event_Manager*, o objeto alvo é localizado na hierarquia dos modelos componentes e seu ponteiro é atribuído ao novo evento. O arcabouço permite desenvolvimento modular e extensível dos modelos componentes e algoritmos da simulação. Assim, modelos componentes, como um modelo matemático, são implementados como especializações das classes bases do arcabouço. Um novo simulador (Controle de Simulação, modelo específico e *SF_Interface*) é construído por meio da especialização das classes *Model* (e suas subclasses) e *SF_Interface*.

A classe abstrata *Model* permite a interação entre suas instâncias e as classes *SF_Interface*, *Control* e *Event_Manager*. Além disto, as instâncias de *Model* contêm as listas de portas de entrada e saída. Para possibilitar a codificação hierárquica dos modelos componentes do usuário, *Model* foi projetada como uma possuidora de instâncias componentes *Model* (isto é, contém uma lista de suas instâncias). Também é responsável pela propagação da chamada da função de transição para suas instâncias na ordem apropriada. Subclasses *Model* suportam paradigmas específicos de modelagem. Por exemplo, as classes *Dyn_Model* e *Dyn_Stock* suportam especificamente modelos de sistemas dinâmicos (Figura 2). Uma instância de uma subclasse *Model* também pode incluir instâncias de outras subclasses *Model* existentes como subsistemas. Instâncias *Model* podem ser conectadas através de instâncias da classe *Port*. A classe *Port* tem um atributo ponteiro para *Data_Source* que fornece os dados da porta. Inicialmente, o ponteiro *data_source* aponta para uma instância de *Variable*, que possui dado do tipo decimal (tipo nativo double da linguagem C++). Uma porta (destino) pode ser conectada a outra porta (fonte) para receber os dados daquela porta. Quando é criada a conexão, o ponteiro *Data_Source* da porta (destino) passa a apontar para a outra porta (fonte) e a partir de então, quando for feita uma operação de leitura será informado o valor da outra porta. Outra possibilidade é conectar a porta a um objeto de *Variable*

ou *Calculated_Variable*, que possui associado uma equação(funçao) para o cálculo da variável.

Um diagrama de sequência *Unified Modeling Language* (UML) de uma execução de simulação sob o arcabouço é mostrado na Figura 2. Toda comunicação entre a aplicação cliente e o simulador é intermediada pela classe *SF_Interface*. O arcabouço reporta erros ou eventos como o início ou fim de uma iteração por meio de *callbacks*. Antes do início de uma simulação, a aplicação cliente precisa registrar as *callbacks* que deseja ser notificada. A aplicação cliente então informa o estado inicial, os nomes e os valores das portas de entrada (variável do modelo). O simulador usa o nome da porta para conectar a porta de entrada apropriada e obter seu valor. Mudanças dos valores das portas de entrada podem ser agendadas como eventos futuros ou atribuídas na *callback* do evento *before_iterate*. A aplicação cliente deve então configurar a simulação (definir o passo de tempo, duração da simulação, etc). A ordem das chamadas da aplicação cliente para preparar a simulação, antes do início da execução, não é importante.

Após informar as entradas e configurar a simulação, a aplicação cliente pode disparar a execução da simulação. O simulador delega a execução da simulação para a classe *Control*. O laço principal da simulação chama a classe *Event_Manager* para processar eventos agendados para acontecer antes do próximo passo de tempo. Estes eventos chamam os métodos apropriados do objeto alvo (por exemplo, *Model* ou *Port*) e são removidos da lista de eventos futuros pelo *Event_Manager*. *Control* pede então à classe *Model* para executar a transição de estado. A transição de estado pode requerer uma série de chamadas à classe *Model*. Como *Model*, na verdade, tem uma estrutura hierárquica, as chamadas de *Control* são propagadas para os modelos componentes sucessivamente até os últimos componentes do nível hierárquico. Embora possa haver uma série de chamadas para instâncias da classe *Model*, todos eles chamam o mesmo método público *Execute* com os argumentos *TargetMethod*, *ArgsVector* e *Propagate*. O argumento *TargetMethod* indica qual o método do modelo deve ser executado. *ArgsVector* passa os argumentos para o método a ser executado (por exemplo, hora e intervalo de tempo para o método *Update*). O argumento *Propagate* indica se o método *Execute* deve ser propagado para outros modelos componentes.

O método *Execute* é responsável por chamar os métodos privados ou protegidos especializados apropriados, dependendo do evento. Depois de tudo isso, o *after_iterate callback* é chamado para deixar o aplicativo cliente consultar o simulador para obter dados de saída, para pausar ou para abortar a simulação.

O uso do arcabouço é exemplificado pela implementação de um modelo dinâmico para o crescimento e a composição de gado proposto por Oltjen et al. (1986). O modelo contém três variáveis de estado que correspondem à massa de proteína do corpo, gordura e DNA. O primeiro passo para o desenvolvimento do simulador para este modelo é a especialização da classe *Dyn_Model* em *Mdl_Oltjen* e da classe *SF_Interface* em *Sim_Oltjen*. O simulador especializado *Sim_Oltjen* é necessário para iniciar o modelo *Mdl_Oltjen* especializado.

Portas de *Mdl_Oltjen* são definidas pela criação de portas nomeadas de entrada e saída usando os métodos *CreateInputs* e *CreateOutputs* (Figura 3). Variáveis de estado (*St_Fat*, *St_Prot*, *St_DNA*) são iniciadas pelo método *CreateComponents*. Variáveis auxiliares são criadas pelo método *CreateVariables* e apontam para funções, por exemplo, *f_DNAMax*. As funções têm de seguir um protótipo definido pelo arcabouço, ou seja, uma função sem parâmetros que retorna um valor de precisão dupla. Funções podem usar os valores das portas de entrada, *Dyn_Stock* ou outras variáveis auxiliares para calcularem seus valores de retorno. Portas de saída estão conectados às portas de saída de *Dyn_Stock* ou a uma variável auxiliar (ver o método *CreateConnections*, Figura 3).

```
class Mdl_Oltjen: public Cls_Dyn_Model{
private:
    Cls_Calculated_Variable*Calc_DNA_Max, *Calc_km, *Calc_d_dt_Fat, …
// calculated variables
    Cls_Dyn_Stock *St_Fat, *St_Prot, *St_DNA; //state variables
    Cls_Port *In_MEI, *In_MEC, *In_Init_Fat, *In_Init_Prot, *In_Init_
DNA; //dynamic input ports
    Cls_Port *InC_LWInit, *InC_CCInit, *InC_MatEBW; //static input
ports – fixed values
    Cls_Port *Out_FatPerc, *Out_BCS, *Out_Maturity, … //output ports
    //methods...};
double Mdl_Oltjen::f_DNAMax(){
  return c_DNARefmax*InC_MatEBW->Value()/c_RefMatEBW;} //equation to
be used by auxiliary variable
void Mdl_Oltjen::CreateVariables(){ //creates the model auxiliary
variables
  Calc_DNA_Max = AddAuxilary("DNA max", this, (dblfunc_ptr_type)&
Mdl_Oltjen::f_DNAMax);
  // other calculated variables}
void Mdl_Oltjen::CreateInputs(){ //creates input ports
  In_MEI = AddInputPort("MEI");
//other input ports}
void Mdl_Oltjen::CreateOutputs(){ //creates output ports
    OutFat = AddOutputPort("Fat", this, Aux_d_dt_Fat);
    //other output ports}
void Mdl_Oltjen::CreateComponents(){
//creates component models of the next lower hierarchy level, if
existing
  St_DNA = new Cls_Dyn_Stock("DNA", this);
  St_DNA->AddInflow("d_dt_DNA");
  St_DNA->SearchInputPort("d_dt_DNA")->BindToDoubleVariable(Aux_d_
dt_DNA);
  AddSubModel(St_DNA);
    //other instances of Cls_Dyn_Stock: St_Fat and St_Prot}
void Mdl_Oltjen::CreateConnections(){//connect ports
    St_DNA->SearchInputPort("InitStock")->dataSource->BindToPort(In_
Init_DNA);
    //other connections }
```

Figura 3. Fragmento de código C++ do modelo Oltjen escrito para o arcabouço.
Fonte: Oltjen et al. (1986).

Resultados e discussão

O código do arcabouço é relativamente pequeno e pode ser aprendido por programadores de graduação iniciantes no projeto em tempo aceitável. Este era um requisito de desenvolvimento, considerando a alta rotatividade de programadores em nosso instituto. A padronização da implementação de modelos também mostrou-se muito importante neste ambiente de alta rotatividade.

Outro resultado alcançado foi o acoplamento fraco entre os componentes do software: a) dentro do simulador, ou seja, entre o controle de simulação e modelos componentes, e b) entre o simulador e outro software - interface gráfica do usuário, pacotes estatísticos, bases de dados, etc. O desacoplamento da descrição textual (ou matemática) do modelo do comportamento dos sistemas e da codificação dos modelos componentes está previsto como trabalho futuro.

O reúso de modelos componentes foi alcançado de forma satisfatória usando o arcabouço. O desenvolvimento de um repositório de modelos agrícolas (como proposto por Bergez et al., 2013) está previsto para o futuro, conforme o número de modelos implementados por nossa equipe cresce.

O arcabouço carece de adesão a um formalismo consolidado (por exemplo DEVS, Zeigler et al., 2000) e estudos futuros do grupo verificarão as vantagens de fazê-lo.

Conclusões

Os autores consideram que o desenvolvimento do arcabouço descrito foi um exercício valioso e útil para apoiar e padronizar o desenvolvimento de modelos de simulação contínua pela equipe, proporcionando autonomia

para desenvolver e customizar as funcionalidades exigidas pelos projetos de simulação específicos.

Os trabalhos em curso incluem o desenvolvimento de uma estrutura flexível para apoiar a implementação e comunicação de algoritmos multicomponentes (ou seja, aqueles que precisam acessar o estado de um conjunto de componentes em uma única simulação, a integração numérica) e algoritmos de multi-simulação (aqueles que precisam executar várias simulações para alcançar um resultado - por exemplo, a otimização heurística, simulação de Monte-Carlo). A padronização de formatos de dados e a comunicação com outro software (R, Excel, Matlab) também está prevista para o futuro próximo.

Referências

BERGEZ, J. E.; CHABRIER, P.; GARY, C.; JEUFFROY, M. H.; MAKOWSKI, D.; QUESNEL, G.; RAMAT, E.; RAYNAL, H.; ROUSSE, N.; WALLACH, D.; DEBAEKE, P.; DURAND, P.; DURU, M.; DURY, J.; FAVERDIN,P.; GASCUEL-ODOUX, C.; GARCIA, F. An open platform to build, evaluate and simulate integrated models of farming and agro-ecosystems. **Environmental Modelling & Software**, Oxford, v. 39, p. 39-49, Jan. 2013.

BOLTE, J. Object-oriented programming for decision systems. In: PEART, R. M.; CURRY, R. B (Ed.). **Agricultural systems modeling and simulation**. New York: Marcel Dekker, c1998. p. 629-650.

BOOTE, K. J.; JONES, J. W.; HOOGEN; BOOM, G. Simulation of crop growth: CROPGRO model. In: **Agricultural systems modeling and simulation**. New York : Marcel Dekker, c1998. p. 651-692.

FILIPPI, J. B.; BISGAMBIGLIA, P. (2003). JDEVS: an implementation of a DEVS based formal framework for environmental modelling. **Environmental Modelling & Software**, Oxford, v. 19, n. 3, p. 261-274, Mar. 2003.

JONES, J. W.; KEATING, B. A.; PORTER, C. H. Approaches to modular model development. **Agricultural systems**, Essex, Inglaterra, v. 70, p. 421-443, Nov./Dec.2001.

MOORE, A. D.; HOLZWORTH, D. P.; HERRMANN, N. I. HUTH, N. I.; ROBERTSON, M. J. The common modelling protocol: a hierarchical framework for simulation of agricultural and environmental systems. **Agricultural Systems**, Essex, Inglaterra, v. 95, p. 37–48, 2007.

NUTARO, J. J. **Building software for simulation**: theory and algorithms, with applications in C++ . Hoboken: .J. Wiley, c2011. 347 p.

OLTJEN, J. W.; BYWATER, A. C.; BALDWIN, R. L.; GARRETT, W. N. Development of a dynamic model of beef cattle growth and composition. **Journal of animal science**, v. 62, p. 86-97, 1986.

QUESNEL, G.; DUBOZ, R.; RAMAT, E.; TRAORÉ, M. K. **VLE**: a multimodeling and simulation environments. In: SUMMER COMPUTER SIMULATION CONFERENCE, 2007, San Diego. **Proceedings**... San Diego: Society for Computer Simulation International, 2007. p. 367-374. SCSC'07.

ZEIGLER, B. P.; PRAEHOFER, H.; KIM, T. G. **Theory of modeling and simulation**: integrating discrete event and continuous complex dynamic systems. San Diego: Academic Press, c2000. 510 p.

Ministério da
Agricultura, Pecuária
e Abastecimento

GOVERNO FEDERAL
BRASIL
PAÍS RICO É PAÍS SEM POBREZA

CGPE 11136