Modelo Genérico de Simulação de Doenças de Plantas em R

Rafael A. Aguiar, Willingthon Pavan, José M. C. Fernandes, Carlos A. Holbig, Jaqson Dalbosco

***Resumo*—Este trabalho apresenta um modelo de simulação genérico para doenças de plantas, desenvolvido na linguagem de programação R, sendo capaz de representar epidemias de doenças de plantas num hospedeiro em crescimento, respondendo à definição parametrizada dessas epidemias e ser executado, concomitantemente, com modelos de simulação de culturas escritos em diferentes linguagens como o Fortran. Apresenta também os padrões existentes para o desenvolvimento Orientado a Objetos em R, e as suas principais distinções. O resultado obtido demonstrou que o modelo genérico pode ser utilizado para simulações em diferentes doenças graças a sua característica de parametrização e dos padrões utilizados em sua implementação, resultado num modelo eficiente e extensível.**

***Palavras Chave*—Modelagem e Simulação, Doenças de Plantas, Modelos Genéricos, Linguagem R.**

I. Introdução

A melhoria dos sistemas de produção agrícola no sul do Brasil é uma tarefa que exige muito esforço. Trata-se de um sistema de produção muito complexo, onde as interações entre fatores do ambiente físico e de ordem socioeconômica acabam determinando o resultado final e condicionando o desempenho do sistema de produção de grãos, requerendo o uso de um enfoque de pesquisa transdisciplinar. Por essa característica, o mais adequado parece ser o uso do enfoque sistêmico, via técnicas de modelagem e simulação e de teorias de decisão [4], integrando conhecimentos e analisando possibilidades que possam otimizar a produção, quer seja no planejamento agrícola [11] ou no manejo das culturas.

O uso de modelos matemáticos aplicados à agronomia vem se expandindo muito rapidamente, destacando-se a modelagem matemática das doenças de plantas [2]. Os recentes avanços na ciência da computação têm auxiliado nesse desenvolvimento, trazendo vantagens e facilidades operacionais para a construção de modelos genéricos que representam os diversos patossistemas. A utilização de modelos que simulam o crescimento e desenvolvimento de uma cultura e, ao mesmo tempo, contabilizam o impacto das doenças pode ser determinante para o auxílio à tomada de decisões. Os cultivos agronômicos apresentam-se como um sistema complexo, afetado por um conjunto de fatores bióticos e abióticos, exercendo grande influência sobre o rendimento.

Os modelos de simulação podem ser vistos como parceiros, fatores contribuintes para o produtor, o qual lhe apresentará informações que possibilitam uma análise para auxílio na tomada de decisão.

Este trabalho tem como objetivo apresentar um modelo genérico para simulação de doenças em plantas, desenvolvido na linguagem de programação R, capaz de representar epidemias num hospedeiro em crescimento, respondendo à definição parametrizada dessas epidemias pré-configuradas e ser executado, concomitantemente, com modelos de simulação de culturas escritos em diferentes linguagens como o Fortran.

A. Modelos de Simulação

Modelos de simulação de culturas, capazes de predizer o rendimento final, têm sido estudados intensivamente em várias partes do mundo e representam a simulação da dinâmica do crescimento das culturas através da integração numérica, com a ajuda de computadores [6].

Os primeiros modelos de simulação de culturas foram desenvolvidos nos anos 60 e tinham como finalidade simular a interceptação da luz e a fotossíntese nas plantas [16]. Com o decorrer do tempo, os modelos tornaram-se mais complexos, necessitando uma maior quantidade de informações para o seu uso. Com a complexidade dos modelos também vieram as limitações, como necessidade de poder de processamento, migração de código para modernas linguagens de programação, entre outras. Mesmo assim, modelos de simulação de culturas tiveram importante participação na pesquisa científica, na tomada de decisão e na educação [2].

Para entender melhor o que são e qual a finalidade dos modelos de simulação, necessita-se compreender a definição de modelos e, após, de simulação. Primeiramente, modelos podem ser definidos como uma representação de algo que se percebe ser realidade, podendo ser um objeto, uma ideia ou até mesmo um sistema. Antes de existir a simulação, é preciso que exista um modelo já definido de algo que se necessita que seja simulado. Portanto, simulação é a técnica de reprodução dos modelos em um processo que reproduza aspectos da realidade.

Os modelos matemáticos de simulação podem ser

Rafael A. Aguiar é graduado em Ciência da Computação pela Universidade de Passo Fundo, Passo Fundo, RS, 99052-900, Brasil (e-mail: rafadeaguiar@gmail.com).

Willingthon Pavan é doutor em Agronomia e professor da Universidade de Passo Fundo, Passo Fundo, RS, 99052-900, Brasil (e-mail: pavan@upf.br).

José M. C. Fernandes é PhD em Fitopatologia e pesquisador da Embrapa Trigo, Passo Fundo, RS 99001-970 Brasil (e-mail: mauricio@cnpt.embrapa.br).

Carlos A. Holbig é doutor em Ciência da Computação e professor da Universidade de Passo Fundo, Passo Fundo, RS, 99052-900, Brasil (e-mail: holbig@upf.br).

Jaqson Dalbosco é mestre em Educação e professor da Universidade de Passo Fundo, Passo Fundo, RS, 99052-900, Brasil (e-mail: jaqson@upf.br).

classificados como [4,15]:

1) **Estáticos ou dinâmicos**: modelos estáticos são aqueles que permitem a descrição do estado do sistema somente para um certo momento, geralmente estes modelos não envolvem a variável tempo. Já os modelos dinâmicos são construídos para representar as alterações de estado do sistema em função do avanço da variável tempo.
2) **Determinístico ou estocástico:** são modelos determinísticos aqueles que em suas formulações não fazem uso de variáveis aleatórias, e não contém elementos probabilísticos. Os modelos estocásticos, por outro lado, possuem pelo menos uma variável de entrada ou parâmetro do sistema tipificado como variável aleatória.
3) **Discretos ou contínuos:** são modelos discretos aqueles em que o avanço da contagem de tempo na simulação é derivado da ocorrência de um evento. Em compensação para os modelos contínuos o avanço do tempo na simulação ocorre de forma contínua com incrementos de valores iguais no tempo.

É notória a existência de um grande número de modelos de simulação desenvolvidos para o setor agrícola, porém, citam-se aqui os principais, de acordo com [2]:

1) **APSIM:** ***A****gricultural* ***P****roduction* ***S****ystems s****IM****ulator*, modelo desenvolvido na Austrália, tendo como objetivo o auxílio aos agricultores e também à agentes de assistência técnica com relação às recomendações de cultivos. Permite que os seus módulos sejam desenvolvidos e inseridos no sistema com grande facilidade, de forma que fiquem distribuídos ao redor do núcleo do simulador. Possui módulos agrossilvopastoris, processos do solo e de gerenciamento.
2) **Daisy:** Modelo agroecológico que simula o crescimento de uma cultura, o balanço de água e de calor, o balanço da matéria orgânica, a dinâmica do amônio e do nitrato, baseado em informações sobre práticas gerenciais e dados climáticos. Permite a construção de complexos cenários para a simulação de diferentes estratégias de gerenciamento e rotação de culturas. Este modelo foi desenvolvido na Dinamarca, e vêm sendo aperfeiçoado desde a década de 80.
3) **Century:** Modelo que têm sido aplicado em diversos ecossistemas e tipos de solo, principalmente sob clima temperado. Possui vários submodelos como o de produção vegetal, água e dinâmica da matéria orgânica do solo. Com os resultados obtidos com o uso deste modelo, fez-se com que seu módulo de simulação de resíduos fosse incorporado à suite do DSSAT, o qual é descrito à seguir.
4) **DSSAT:** ***D****ecision* ***S****upport* ***S****ystem for* ***A****grotechnology* ***T****ransfer***,** uma das mais conhecidas aplicações de modelos para simulação do crescimento e desenvolvimento de culturas, sendo desenvolvido pelo projeto IBSNAT (*International Benchmark Sites Network for Agrotechnology Transfer*) e mantido até meados de 2011 pelo consórcio ICASA (*International Consortium for Agricultural Systems Applications*), sendo continuado por membros voluntários, comunicando-se por meio de servidores de listas. O DSSAT é um sistema que envolve diversos modelos de simulação, sendo utilizado em estudos a longo prazo para avaliar estratégias eficientes no gerenciamento da cultura, e que também otimizem a sua produção, informações que são obtidas combinando dados de solo e de clima com modelos de culturas e aplicativos, com o objetivo de simular resultados de vários anos de estratégias no manejo de culturas [3].

B. Importância do uso de Orientação a Objetos

A orientação a objetos se tornou muito popular e difundida por trazer benefícios para o desenvolvimento de sistemas complexos, principalmente por questões como extensibilidade, que é a capacidade do sistema em se desenvolver pelo acréscimo de novos componentes, e reusabilidade, onde um trecho de código é desenvolvido de forma a ser útil a outros [5].

Paradigma de programação no qual um programa é estruturado por meio de objetos, enfatizando aspectos como abstração, encapsulamento, polimorfismo e herança. Ao contrário da ideia de se utilizar funções independentes que serão utilizadas em conjunto, divide-se conceitualmente o 'problema' em partes (objetos), os quais terão atributos que os descrevem, tornando os objetos o mais próximo possível da realidade, tanto do seu funcionamento (métodos), como a sua descrição (atributos).

Considerando que os modelos de simulação pertencem à classe dos sistemas de software, é natural que seja aplicado a eles os conceitos de orientação a objetos durante o desenvolvimento. Porém, é preciso que seja feita uma distinção entre paradigma de simulação, ou seja, recursos utilizados na construção de um modelo, e um paradigma de projeto e implementação aplicado ao desenvolvimento de sistemas de simulação, que utilizam conceitos de modelagem distintos [5].

O desenvolvimento de modelos com este paradigma surgiu juntamente com a primeira linguagem construída para incorporar os conceitos de orientação a objetos, a linguagem Simula. Com isso, fica claro que desde aquela época (década de 60) já existia uma tendência em construir modelos, que pudessem ser um reflexo da realidade. Considera-se então, que modelos de simulação são uma coleção de entidades que interagem entre si, incorporando conceitos de orientação a objetos [5].

C. Problema de Pesquisa

Quando se fala em simulação, é preciso que se tenha em mente a necessidade de desenvolver um sistema buscando uma melhor performance de execução, além de resultados exatos. Atualmente, grande parte dos modelos de simulação, estão escritos em Fortran [2], uma das linguagens mais antigas ainda utilizadas, e que possui uma excelente velocidade de execução.

Modelos de doenças de plantas têm sido utilizados para o

manejo e previsão de epidemias, realizando projeções futuras quanto ao desenvolvimento das doenças [10]. O modelo de doença sobre o qual este trabalho se baseou possui estas características, sendo originalmente desenvolvido na linguagem de programação Java, podendo ser integrado com modelos de crescimento de culturas em linguagens como o Fortran, por exemplo, mas para isto é necessário que haja um acoplamento entre estes.

Alguns aspectos importantes precisam ser considerados para que exista uma integração entre modelos. Pode-se destacar, por exemplo, a identificação de modelos apropriados, especificação das interações (conversões de dados, tipos de variáveis etc), e a verificação da possibilidade de integração dos códigos fontes [7].

Uma das soluções existentes para integração de modelos em Java com Fortran é a comunicação via TCP/IP, onde pode ser citado o servidor Rserve, desenvolvido por Simon Urbanek sob licença GPL [14,9], realizando a comunicação entre as linguagens de programação. O modelo desenvolvido por [2] utilizou este servidor para a integração entre os modelos, entretanto existe uma perda de tempo em execução e complexidade de desenvolvimento, além da necessidade de adequação dos modelos legados para que se usufrua desta tecnologia.

Decorrente do problema citado nota-se que é de grande valia o desenvolvimento de um modelo genérico desenvolvido na linguagem R, possibilitando o fácil acoplamento entre modelos, haja vista a possibilidade de incorporação de código Fortran na linguagem R.

Na seção 2, são apresentadas informações introdutórias do funcionamento da Linguagem R, e os padrões existentes para desenvolvimento Orientado a Objetos para esta linguagem. Após estas noções e informações, são demonstrados os padrões escolhidos para o desenvolvimento do modelo, bem como o seu funcionamento.

II. Implementando uma solução em R

A. Linguagem R

O R é uma linguagem de programação, além de um ambiente integrado para realização de cálculos estatísticos e geração de gráficos complexos. É um projeto de software livre, disponível sob os termos da licença da Fundação Software Livre GNU GPL (*General Public License*). Encontra-se disponível em diferentes plataformas como Linux, Windows e MacOS.

O R é muito similar à linguagem S, que influenciou o seu desenvolvimento e, ainda hoje possui códigos escritos que rodam inalterados no R. Caracteriza-se por ser altamente expansível, através dos seus pacotes, que são bibliotecas desenvolvidas para algumas funções específicas, podendo o desenvolvedor criar novos pacotes e submetê-los à rede de distribuição do R, conhecida por CRAN (*Comprehensive R Archive Network*), onde será analisado e posteriormente disponibilizado na rede.

Como citado anteriormente, um dos pontos fortes do R, é a facilidade com que gráficos de alta qualidade podem ser produzidos. Uma linguagem que foi desenvolvida de forma a ser simples e eficaz, possuindo características como estruturas de controle, funções definidas pelo usuário (programação estruturada e modular), recursividade, facilidade de manipulação de arquivos e programas externos, orientação a objetos, entre outras. Relacionam-se, dessa forma, algumas vantagens ao utilizar o R [8]:

1) Aceita diversas fontes de entrada de dados, como arquivos txt, banco de dados, arquivos CSV, entre outras;
2) Modularidade, por implementar um conceito de dividir o programa, ou software em módulos para uma melhor legibilidade e manutenabilidade;
3) Gera informações de saída para: banco de dados, gráficos, arquivos CSV, e até mesmo gráficos interpolados para o Google Maps;
4) Portabilidade, pois permite a sua execução na mais diferentes plataformas;
5) Orientação a Objetos.

B. Padrões de desenvolvimento Orientado a Objetos em R

A manipulação de dados na linguagem R se dá totalmente na forma de objetos, podendo-se dizer que tudo em R é um objeto [13]. Atualmente, existem três formas diferentes de implementar programação orientada a objetos nesta linguagem, sendo descritas a seguir.

1) Sistema S3

Uma implementação mais antiga, sendo integrada ao R desde a sua formulação. Para utilizá-la é necessário a carga do pacote R.oo [1]. Considerado um sistema informal e simples, ao contrário de linguagens como Java e C++ onde a computação consiste em invocar métodos dos objetos, no S3 a computação é baseada totalmente em funções. Com isso, um método não pertence a uma classe, mas sim a uma função genérica a qual foi definido [13]. Para este sistema, pode-se dizer que as classes e os métodos não são formalmente definidos. O que torna o sistema simples, e também sujeito a falhas [13].

Em S3 a criação de uma nova classe consiste basicamente em definir uma função (chamada de construtora), a qual é utilizada para criar os objetos da classe, representada na Figura 1.

```
1 setConstructorS3("Weather", function(tMin=0,tMax=0){
2                     extend(Object(), "Weather", tMin=tMin, tMax=tMax);
3                   })
```

Fig. 1. Declaração de classe em S3

A Figura 1, acima, ilustra a declaração da classe *Weather*, a qual possui dois campos tMin, e tMax inicializados com o valor 0. Ressaltando-se que toda classe declarada com este padrão necessita estender a classe *Object*, como pode ser analisado na linha 2.

As Figuras 2 e 3, apresentam a declaração dos métodos,

tendo como argumentos da função *setMethodS3*, o nome do método, da classe à que ele pertence e a declaração da função (*function*) onde é descrito todo o seu funcionamento, possibilitando também a adição de parâmetros para a função (Figura 3).

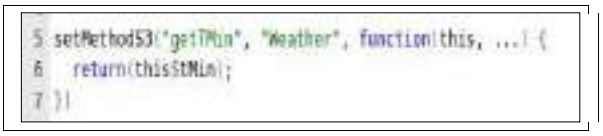

Fig. 2. Declaração de um método em S3

Fig. 3. Declaração de um método com parâmetro em S3

Para instanciar um objeto da classe declarada na Figura 1, e através desta acessar os seus métodos representados nas Figuras 2 e 3, é necessário que se execute os comandos apresentados na Figura 4 (criação do objeto, linha 15, e acesso aos seus métodos, linhas 16 e 17).

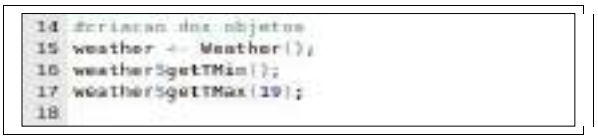

Fig. 4. Criação de um objeto S3 e acesso a seus métodos

2) Sistema S4

Este sistema está disponível de forma mais estável, desde a versão 1.7.0, ou seja, surgiu após o S3. As suas classes são definidas de forma explícita pelo comando *setClass*. Este padrão possui um grau de formalismo maior em comparação ao S3, obrigando a declaração formal dos métodos de uma classe, ao invés da convenção informal do sistema S3 [13], o que trás mais exigências ao desenvolvedor.

Baseia-se na mesma idéia de os objetos terem classes e na existência de funções genéricas que depois invocam métodos específicos para cada tipo de objeto, dependendo da sua classe [13]. Na Figura 5 é ilustrada uma declaração de classe no sistema S4.

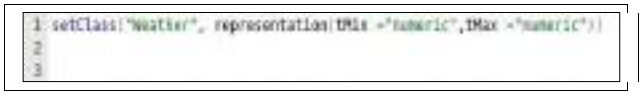

Fig. 5. Declaração de classe em S4

A função *setClass*, linha 1 da Figura 5, permite definir uma nova classe a qual terá o seu nome definido pelo primeiro argumento, neste caso a classe *Weather*. No segundo argumento é indicado uma representação da classe, que é obtida por uma chamada à função *representation*, que recebe como argumentos os nomes dos *slots*, que seriam os atributos da classe. Após, para que esta nova classe estenda de outra, pode-se adicionar a propriedade *contains*, com o nome da classe que se deseja estender. Importante ressaltar que é necessário identificar o tipo do atributo criado para a classe.

Como já citado nesta seção, o padrão S4 possui um grau de formalismo maior que o seu antecessor S3, desta forma, antes da declaração dos métodos para a classe, é necessário inicialmente declarar funções genéricas, para que depois se possa utilizá-las para criação dos métodos, como é apresentado na Figura 6.

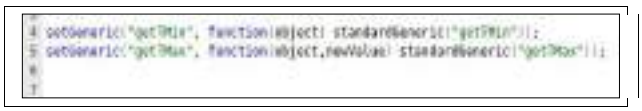

Fig. 6. Declaração de funções genéricas

As Figuras 7 e 8 apresentam as formas de declaração para métodos da classe representada na Figura 5. Nota-se que para este sistema é utilizado o comando *setMethod* para identificar que será descrito um método, onde seu primeiro argumento é o seu nome, após é a classe à que será vinculado, e por fim a escrita da função que será executada.

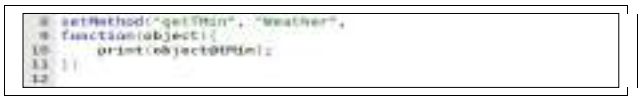

Fig. 7. Declaração de um método com um parâmetro em S4

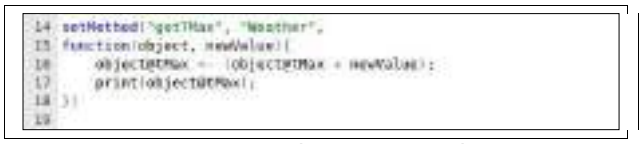

Fig. 8. Declaração de um método com dois parâmetros em S4

Para se obter um objeto a partir desta classe, é necessário executar os comandos apresentados na Figura 9, ressaltando-se que na linha 21 cria-se um objeto da classe *Weather* e inicializa-se seus atributos com os valores 5 e 27. Nas linhas 22 e 23 são executados os métodos desenvolvidos e apresentados anteriormente.

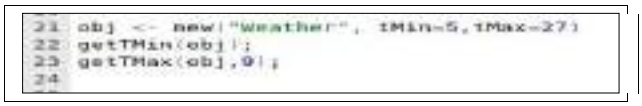

Fig. 9. Criação de um objeto S4 e acesso a seus métodos

3) Sistema R5

Este é o novo padrão de desenvolvimento orientado a objetos em R, implementado a partir da versão 2.12. Ainda está sendo aprimorado, porém já possui uma estrutura sólida. A principal diferença entre este sistema e os apresentados anteriormente, é que o R5 utiliza transmissão de mensagens entre seus objetos, além de serem mutáveis (objetos cujo estado pode mudar). Estas características tornam o funcionamento deste sistema parecido com o das linguagens Java e C# [12]. A implementação deste sistema, foi inteiramente desenvolvido na própria linguagem R, sendo combinações de métodos do sistema S4 e de funções do ambiente [12]. A Figura 10 ilustra a declaração de uma classe, e os seus respectivos métodos no padrão R5.

```
1  setRefClass("Weather",
2       fields = list(tMin="numeric",tMax="numeric"),
3
4       methods = list(
5          initialize = function() {
6               tMin <<- 8;
7               tMax <<- 31;
8
9               callSuper();
10         },
11         getTMin = function() {
12            return(tMin);
13         },
14         getTMax = function(newValue) {
15            tMax <<- (tMax + newValue);
16            return(tMax);
17         }
18    ))
```

Fig. 10. Declaração de classe e métodos em R5

Analisando a Figura 10, identifica-se na linha 1 o comando *setRefClass* responsável por declarar a classe, atributos e também os seus métodos. O primeiro argumento deste comando, é o nome que será atribuído à classe, o próximo será a propriedade *fields* (Figura 10, linha 2) que recebe uma lista com todos os campos necessários, podendo-se utilizar a propriedade *contains* (utilizada para casos em que se deseja estender outra classe), a seguir será realizado a declaração dos métodos pela propriedade *methods* (Figura 10, linha 4), também recebendo uma lista com todos os métodos descritos para este.

Quanto à declaração de métodos é importante ressaltar dois aspectos que podem ser verificados na Figura 10, acima. Primeiro é a função *initialize*, portando-se como construtor da classe, sempre executado quando se cria um novo objeto. Segundo é a diferenciação de acesso às variáveis locais (visibilidade apenas no método a qual foi declarado) e de instância (visível a todos os métodos da classe). Sua diferenciação se dá pela utilização dos símbolos "<<-", para variáveis de instância, e "<-" para variáveis locais.

A Figura 11 apresenta os comandos necessários para se instanciar um novo objeto e acessar os seus métodos, forma semelhante ao sistema S4.

```
23 obj <- new("Weather")
24 obj$getTMin()
25 obj$getTMax(10)
26
```

Fig. 11. Instância objeto R5

C. Funcionamento do Modelo

O modelo genérico aqui apresentado teve seu desenvolvimento baseado na estrutura proposta na tese de doutorado de Pavan (2007), o qual projetou um modelo parametrizado na linguagem de programação Java.

O modelo é composto por módulos para melhor descrever o seu funcionamento, descritos como *Simulator* (Simulador), *InitialCondition* (Condições Iniciais), *Plant* (Planta), *Organ* (Orgão), *Disease* (Doença), *Cloud* (Núvem), *CloudO* (Núvem do Órgão), *CloudP* (Núvem da Planta), *CloudF* (Núvem do Campo), *LesionCohort* (Grupos de Lesões), *GenericR* (Controlador do Modelo), *RIntegration* (Integração com Modelos), *Basic* (Interface implementada pelos demais módulos), *Weather* (Clima), desenvolvidos como classes do padrão R5 e S3 na linguagem R, os quais descrevem as características do grupo que estão representando, organizados de forma hierárquica a fim de representar, de uma forma mais próxima possível da realidade, o modelo.

O modelo caracteriza-se como sendo de passo diário, alimentando-se de dados climáticos, tanto prognósticos como observados ou capturados de estações meteorológicas. Cada componente do modelo é entendido como um objeto ou um grupo de objetos, ou seja, uma entidade virtual que representa as características essenciais da entidade real [2].

Na Figura 12 é apresentado o diagrama de classes para os objetos do modelo, sendo descritos de forma detalhada para cada módulo.

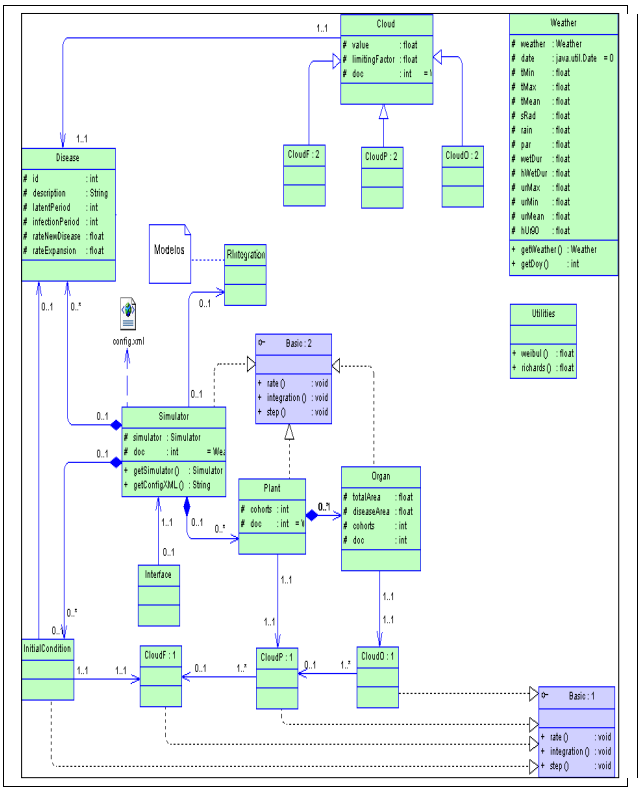

Fig. 12. Diagrama de classes do modelo genérico

1) Simulator

É o módulo responsável pelo controle do fluxo de toda simulação, podendo ser independente ou controlado pelo módulo GenericoR, o qual pode determinar a velocidade da execução e solicitar os dados de um determinado instante da simulação, a fim de que possam ser visualizados e analisados (dados de um determinado "dia" da simulação). Possui uma

estrutura básica, como todos os demais módulos, com "inicialização", "taxa", "integração" e "saída". Ao ser inicializado é feito uma leitura dos parâmetros do modelo (config.xml), que possui informações como tipos e quantidades de doenças a simular, configurações do processo de simulação, quantidade de plantas a simular, datas de semeadura, entre outras.

2) Rintegration

É o módulo responsável por se comunicar com outros modelos, seja de forma direta ou buscando informações. Com este, pode ser feito integrações com modelos de crescimento e desenvolvimento de plantas como o CropSim-Wheat, desenvolvido em Fortran e que pertence à suite do DSSAT.

3) Weather

O módulo *Weather* é o responsável por alimentar os sistemas de simulação com dados climáticos, advindos tanto de estações meteorológicas como de prognósticos climáticos. Disponibiliza aos demais módulos informações como temperatura e precipitação até formas de controle de navegação (passar para o próximo dia, por exemplo). Estas informações climáticas podem estar tanto em bases de dados relacionais, arquivos texto ou arquivos em formato XML (*eXtended Markup Language*).

4) Utilities

Este módulo realiza os cálculos dos diversos tipos de equações, os quais podem ser citados: *trapezoidalFunction*, *temperatureFavorability*, *wetnessFavorability*, entre outros.

5) InitialCondition

Módulo responsável por representar as condições iniciais de inóculo, determinando o momento em que há liberação dos primeiros esporos do patógeno no ambiente.

6) Cloud, CloudF, CloudP e CloudO

Estes módulos são responsáveis por representar a quantidade de esporos disponíveis para novas infecções e são divididos a fim de estimar a concentração de esporos e, consequentemente de novas infecções em três escalas organizacionais: campo (*CloudF*), planta (*CloudP*), e órgão (*CloudO*). O objetivo é estruturar a epidemia numa hierarquia espacial de deposição de esporos: auto deposição, deposição de um propágulo produzido numa lesão localizada na própria folha e alo-deposição, deposição de um propágulo produzido numa lesão localizada em outra folha da mesma planta ou em outra planta.

7) Disease

Este módulo serve como base de informação sobre as características da doença, o qual possui métodos de acesso a tais características. Além dos métodos de acesso aos dados, há um método para o cálculo de novas lesões.

8) Plant

É responsável por representar a planta, com seus órgãos e suas respectivas áreas (sadias e infectadas); possui uma ligação forte com o módulo RIntegration, em virtude da busca, a cada passo da simulação, de dados sobre o desenvolvimento da planta, atualizando seus respectivos dados, além de retroalimentar o simulador com dados sobre áreas infectadas.

9) Organ

É responsável por representar os órgãos de uma planta, verificar o avanço da epidemia em seus três estádios (latente, infeccioso e necrótico), computar sua área sadia, área infectada, área necrótica e senescente. Na inicialização deste módulo é criado um objeto do módulo *CloudO*, um para cada tipo de doença, além de buscar do módulo *Rintegration* a sua área atual.

10) LesionCohort

Este é considerado um dos módulos mais importantes, pois é o responsável pelo processo epidemiológico, contribuindo ou não para a propagação da doença [2].

D. Solução implementada em R

Para sua execução é necessário a carga dos pacotes R.oo, e XML, além da versão 2.12 ou superior do R. O desenvolvimento deste modelo, baseado nos módulos já citados, se fez uso de dois sistemas existentes em R para a programação orientada a objetos, são eles: R5 e S3. O uso dos dois padrões se deve pelo fato da necessidade em representar classes no padrão *Singleton* (padrão que garante a existência de apenas uma instância da classe, havendo assim apenas um ponto de acesso global à ela no sistema). Com isto, utilizaram-se classes no padrão S3 para armazenar tais instâncias do objeto, controlando para que todos os demais acessem esta mesma instância.

As classes *Config*, *Disease*, *Rintegration*, *Simulator*, *Utilities*, e *Weather* não possuem acesso direto pelas demais classes, pois utilizam a mesma instância durante toda execução. A Figura 13 apresenta de forma simplificada o comportamento dos objetos dessas classes do modelo.

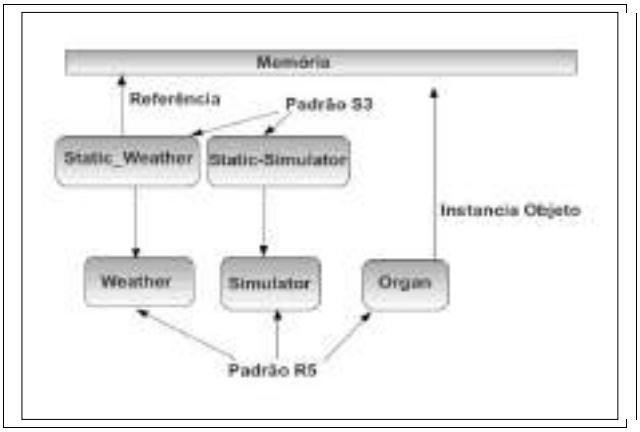

Fig. 13. Exemplo básico da estrutura das classes

A Figura 14 apresenta a classe *Static_Weather* utilizada para acessos ao objeto *Weather*, retornado pela chamada ao método *getInstance* (linha 11).

Fig. 14. Classe *Static_Weather*, padrão S3

A Figura 15 representa uma parte do desenvolvimento da classe *Weather* (padrão R5). Na linha 1, pode-se verificar que toda declaração da classe é atribuída a uma variável com mesmo nome, servindo para que exista uma referência em memória. Sendo assim, ao invés de instanciar um objeto com ***new('Weather')***, se utiliza ***Weather$new()***.

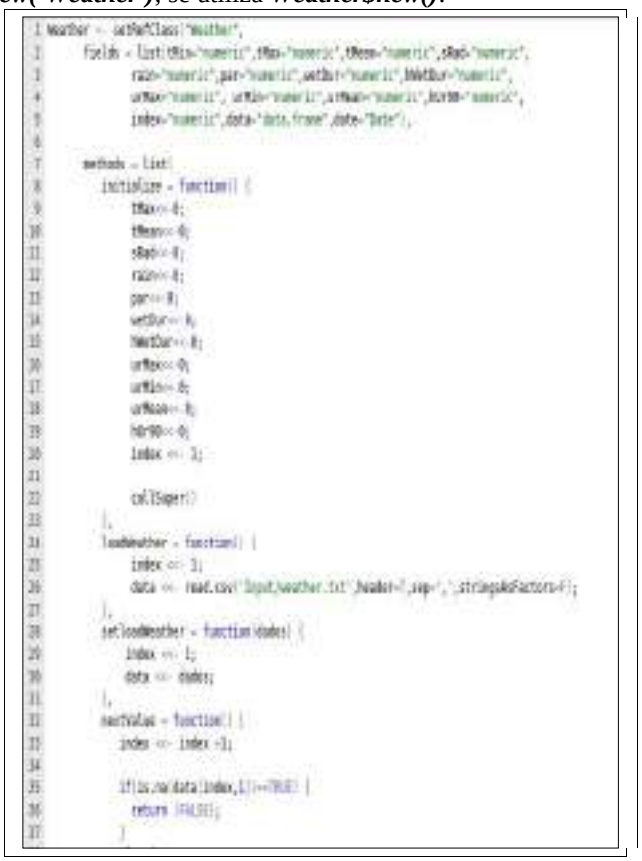

Fig. 15. Classe *Weather*, padrão R5

Para acesso à classe *Weather*, é executado comando ***Static_Weather$getInstance(),*** obtendo-se um objeto da classe *Weather*.

Toda execução do modelo se dá através da classe *GenericoR*, a qual inicializa e controla os passos do modelo, sendo o código da classe apresentado na Figura 16.

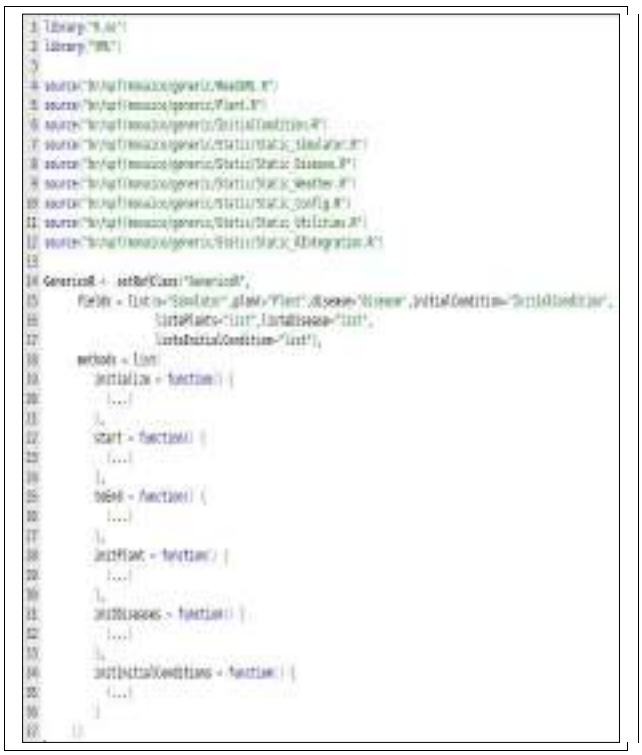

Fig. 16. Classe *GenericoR*, padrão R5

A estrutura de pastas do modelo é apresentada na Figura 17, destacando-se a pasta Input, onde se encontram os arquivos inicialmente necessários para execução do modelo, sendo representados na Figura 18.

Fig. 17. Estrutura de pastas do modelo

Fig. 18. Arquivos da pasta Input, configurações iniciais

Conforme a Figura 18, verifica-se que o arquivo config.xml é o responsável por parametrizar as epidemias que serão simuladas, enquanto o arquivo weather.txt armazena os dados meteorológicos analisados para o tempo determinado que deseja-se realizar a simulação. A Figura 19 apresenta o conteúdo do arquivo config.xml, seguida pela Figura 20 com o conteúdo do arquivo weather.txt.

Fig. 19. Conteúdo arquivo config.xml

```
DATA,TMIN,TMAX,TMEAN,SRAD,RAIN,HUR90
01/01/2004,10.6,23.4,17,29.5,0.4,0
01/02/2004,12.4,26.6,19.5,28,0,3
01/03/2004,14.5,27.3,20.9,28.8,0,0
01/04/2004,14,26.9,20.45,29,0,0
01/05/2004,15.3,27.8,21.55,22.6,0,0
01/06/2004,18.5,31,24.75,22,8,5
01/07/2004,17.3,29,23.15,19.2,10.1,7.5
01/08/2004,18.6,27.8,23.2,18.6,1.8,11.5
01/09/2004,17.5,30.9,24.2,25,16,8.3
01/10/2004,17.9,31,24.45,24.5,0,2
01/11/2004,18.6,29.8,24.2,22.3,0,5.8
01/12/2004,14.8,26.2,20.5,24.3,6.3,8
01/13/2004,16.7,27.5,22.1,21.5,0,3
01/14/2004,16.8,26.3,21.55,14.2,3.6,10.3
01/15/2004,15.3,28.2,21.75,26.8,2,7.5
01/16/2004,15.9,27.9,21.9,23.5,0,0
```

Fig. 20. Conteúdo arquivo weather.txt

Para executar o modelo é necessário apenas a execução do arquivo *GenericoR* seguido dos comandos:

1) genericoR <- GenericoR$new()
2) genericoR$toEnd()

Com o término da execução dos comandos acima, será gerado um arquivo de saída contendo os dados finais da simulação, conforme ilustrado na Figura 21.

Fig. 21. Dados gerados como resultado para a simulação

III. CONCLUSÃO

O modelo proposto e implementado, correspondeu as expectativas, mostrando-se capaz de responder, genericamente, aos parâmetros das doenças especificadas sob sua estrutura parametrizada e modular, representando as epidemias programadas, não só sob o aspecto da doença mas como o conjunto patógeno/hospedeiro.

A utilização da linguagem R para desenvolvimento orientado a objetos foi satisfatória. Usando os sistemas R5 e S3 conseguiu-se fazer proveito de técnicas como herança, troca de mensagens entre objetos e classes *Singleton*, o que tornou o desenvolvimento do modelo próximo de outras linguagens como Java, e C#.

REFERÊNCIAS

[1] H. Bengtsson, "The R.oo package - Object-Oriented Programming with References Using Standard R Code", In Kurt Hornik, Friedrich Leisch and Achim Zeileis, editors, Proceedings of the *3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, March 20-22, Vienna, Austria. http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/

[2] A. Graves, et al. "Crop simulation models as tools in computer laboratory and classroom-based education". *Journal of Natural Resources and Life Sciences Education*, v. 31, p. 48–54, 2002.

[3] Icasa. (2011, 16 Junho). *ICASA - International Consortium for Agricultural Systems Applications*. 2011. Disponível em: http://icasa.net.

[4] A. M. Law, W. D. Kelton, *Simulation modeling & analysis*, 2nd Edition, McGraw Hill, Inc: New York. 1991.

[5] B. A. Mello. (2011, 15 Novembro). *Modelagem e Simulação de Sistemas*, 2001. Disponível em: http://pt.scribd.com/doc/50045916/30/Simulacao-orientada-a-objetos.

[6] W. Pavan, "Técnicas de engenharia de software aplicadas à modelagem e simulação de doenças de plantas". 2007. 182 p. Tese (Doutorado em Agronomia) - Universidade de Passo Fundo, 2007.

[7] J. E. Pedrini, *Abordagens computacionais para o acoplamento de modelos*, 2009.

[8] *R Project*. (2011, 05 Março). Disponível em: http://www.r-project.org.

[9] Rserve. (2011. 15 Março.) *About Rserve*. 2011. Disponível em: http://www.rforge.net/Rserve.
[10] R. SEEM, "Plant disease forecasting in the era of information technology". In: *Plant Disease Forecast: Information Technology in Plant Pathology*. Kyongju, Republic of Korea: [s.n.], 2001.
[11] L. C. Silva, "Stochastic simulation of the dynamic behavior of grain storage facility". 2002. Tese (Doutorado em Engenharia Agrícola.) Universidade Federal de Viçosa. Viçosa: MG.
[12] Github Social Coding. (2011, 10 Abril). "The R5 Object System". Disponível em: https://github.com/hadley/devtools/wiki/R5.
[13] L. Torgo, *A Linguagem R Programação para a análise de dados*, 2009 – Escolar Editora.
[14] S. Urbanek. (2011, 15 Março). "Rserve: A fast way to provide R functionality to applications". 2003. Disponível em: http://www.ci.tuwien.ac.at/Conferences/DS2003/Drafts/Urbanek.pdf.
[15] L. Willocquet, S. Savary, "An epidemiological simulation model with three scales of spatial hierarchy". *Analytical and Theoretical Plant Pathology*, The American Phytopathological Society, v. 94, n. 8, p. 883-891, 2004.
[16] C. de Wit, "Photosynthesis of leaf canopies". *Agric. Res. Rep., Pudoc*, Wageningen, the Netherlands, v. 663, 1965.