UNIVERSIDADE DE PASSO FUNDO

INSTITUTO DE CIÊNCIAS EXATAS E GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

Um Middleware em C++: Modernizando a interface de modelos de cultivos codificados em Fortran

Felipe de Vargas

Passo Fundo

2017

UNIVERSIDADE DE PASSO FUNDO

INSTITUTO DE CIÊNCIAS EXATAS E GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

UM MIDDLEWARE EM C++: MODERNIZANDO A INTERFACE DE MODELOS DE CULTIVOS CODIFICADOS EM FORTRAN

Felipe de Vargas

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Computação Aplicada na Universidade de Passo Fundo.

Orientador: Prof. Carlos Amaral Hölbig

Coorientador: Prof. Willingthon Pavan

Passo Fundo

2017

CIP – Catalogação na Publicação

V297m Vargas, Felipe de
Um Middleware em C++ : modernizando a interface de modelos de cultivos codificados em Fortran / Felipe de Vargas. – 2017.
70 f. : il. color. ; 30 cm.

Orientador: Prof. Dr. Carlos Amaral Hölbig.
Coorientador: Prof. Dr. Willingthon Pavan.
Dissertação (Mestrado em Computação Aplicada) – Universidade de Passo Fundo, 2017.

1. Simulação (Computadores). 2. Middleware. 3. Arquitetura de software. 4. Interface de programas aplicativos (Software). I. Hölbig, Carlos Amaral, orientador. II. Pavan, Willingthon, coorientador. III. Título.

CDU: 004.4

Catalogação: Bibliotecário Luís Diego Dias de S. da Silva – CRB 10/2241

UPF Universidade de Passo Fundo | PPGCA Programa de Pós-Graduação em Computação Aplicada Instituto de Ciências Exatas e Geociências - ICEG

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO DO ACADÊMICO

FELIPE DE VARGAS

Aos vinte e quatro dias do mês de março do ano de dois mil e dezessete, às 15 horas, realizou-se, no Instituto de Ciências Exatas e Geociências, prédio B5, da Universidade de Passo Fundo, a sessão pública de defesa do Trabalho de Conclusão de Curso **"Um Middleware em C++: modernizando a interface de modelos codificados em Fortran"**, de autoria de Felipe de Vargas, acadêmico do Curso de Mestrado em Computação Aplicada do Programa de Pós-Graduação em Computação Aplicada – PPGCA/UPF. Segundo as informações prestadas pelo Conselho de Pós-Graduação e constantes nos arquivos da Secretaria do PPGCA, o aluno preencheu os requisitos necessários para submeter seu trabalho à avaliação. A banca examinadora foi composta pelos doutores Carlos Amaral Hölbig, Willingthon Pavan, José Maurício Cunha Fernandes e Diego Noleto Luz Pequeno. Concluídos os trabalhos de apresentação e arguição, a banca examinadora considerou o candidato APROVADO. Foi concedido o prazo de até quarenta e cinco (45) dias, conforme Regimento do PPGCA, para o acadêmico apresentar ao Conselho de Pós-Graduação o trabalho em sua redação definitiva, a fim de que sejam feitos os encaminhamentos necessários à emissão do Diploma de Mestre em Computação Aplicada. Para constar, foi lavrada a presente ata, que vai assinada pelos membros da banca examinadora e pela Coordenação do PPGCA.

Prof. Dr. Carlos Amaral Hölbig
Presidente da Banca Examinadora
(Orientador)

Willingthon Pavan

Prof. Dr. Willingthon Pavan
(Coorientador)

Prof. Dr. José Maurício Cunha Fernandes
(Avaliador Interno)

Diego Noleto Luz Pequeno

Prof. Dr. Diego Noleto Luz Pequeno
(Avaliador Externo)

Prof. Dr. Rafael Rieder
Coordenador do PPGCA

“Seu trabalho vai preencher uma parte grande da sua vida, e a única maneira de ficar realmente satisfeito é fazer o que você acredita ser um ótimo trabalho. E a única maneira de fazer um excelente trabalho é amar o que você faz.”

(Steve Jobs)

UM MIDDLEWARE EM C++: MODERNIZANDO A INTERFACE DE MODELOS DE CULTIVOS CODIFICADOS EM FORTRAN

RESUMO

Os modelos de simulação de culturas tem sido utilizados com grande sucesso em todo o mundo para aumentar a renda agrícola e reduzir os custos na produção e os recursos humanos necessários para analisar e tomar decisões complexas. Mas está ficando cada vez mais difícil expandir e melhorar esses modelos devido à sua complexidade, tornando-se uma tarefa árdua e cara. A maioria dos modelos são/foram desenvolvidos em Fortran, linguagem que tem limitações na integração, na interoperabilidade, na visualização de dados e no acoplamento dos modelos. O DSSAT-CSM é um sistema de apoio à tomada de decisão, desenvolvido para facilitar a aplicação de modelos de culturas numa abordagem sistemática nas pesquisas agronômicas composto por mais de 42 modelos. Este trabalho visa a criação de dois *middlewares* em C++ para modelos desenvolvidos em Fortran. Um *middleware* que permite a remoção completa das dependências com arquivos texto, tanto para entrada quanto para a saída, permitindo integração do Fortran com outras linguagens de programação. Para provar a eficiência do *middleware* desenvolveu-se um pacote R chamado RsimpleCrop utilizando o modelo genérico chamado de SimpleCrop. O R é responsável por realizar a aquisição dos dados de entrada e passa-lós como parâmetro para o *middleware*, o qual irá manter os dados memória para uso do modelo, após o termino da execução, o modelo retorna para o *middleware* os resultados que serão devolvidos para o pacote permitindo com que o usuário manipule-os como objetos R. O outro *middleware* desenvolvido tem o objetivo de gerenciar os dados de entrada armazenando-os em memória, através de funções genéricas que sejam configuradas em tempo de execução e as saídas geradas pelo DSSAT-CSM, permitindo com que o usuário formate as saídas em tempo de execução, removendo toda a manipulação dos arquivos do Fortran reduzindo, assim, a quantidade de chamadas de sistema para manipular arquivos. Permitindo também, novos formatos de entrada e saída, como o Json, o CSV, dentre outros. Também permite a integração do DSSAT-CSM com sistemas de informação, outras formas de visualização dos dados, facilitar o acoplamento com outros modelos.

Palavras-Chave: DSSAT-CSM, middleware, RSimpleCrop, interface, Json, CSV, pacote R.

A C++ MIDDLEWARE:IMPROVEMENT OF FORTRAN CODED CROP MODELS

ABSTRACT

Crop models have been used with great success worldwide to increase farm incomes and reduce production costs and the human resources needed to analyze and make complex decisions. But it is becoming increasingly difficult to expand and improve these models because of their complexity, making it a hard and expensive task. Most of the models are / have been developed in Fortran, a language that has limitations on integration in interoperability, data visualization and coupling of models. The DSSAT-CSM is a decision support system developed to facilitate the application of crop models in a systematic approach in agronomic research composed of more than 42 models. This work aims to create two middlewares in C ++ for models developed in Fortran. A middleware that allows the complete removal of dependencies with text files, both for input and output, allowing integration of Fortran with other programming languages. To prove the efficiency of the middleware, an R package called RsimpleCrop was developed using the generic model called SimpleCrop. OR is responsible for performing the acquisition of the input data and passes it as parameter to the middleware, which will keep the data memory for use of the model, after the completion of the execution, the model returns to the middleware the results that will be returned For the package allowing the user to manipulate them as R objects. The other middleware developed has the goal of managing the input data by storing them in memory, through generic functions that are configured at runtime and the outputs generated by the DSSAT-CSM, allowing the user to format outputs at runtime, removing all manipulation of Fortran files, thereby reducing the amount of system calls to manipulate files. It also allows new input and output formats, such as Json, CSV, among others. It also allows the integration of DSSAT-CSM with information systems, other forms of visualization of the data, facilitate the coupling with other models.

Keywords: DSSAT-CSM, middleware, RSimpleCrop, interface, Json, CSV, package R.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Os modelos de simulação são representações matemáticas de sistema do mundo real, usados para reduzir tempo e custo da experimentação em campo. São aplicados nas mais diversas áreas de conhecimento, como, por exemplo, ciências sociais e humanas, ciências exatas, dentre outras e tem como objetivo o auxílio na compreensão e prevenção de fenômenos complexos [1]. Diversos modelos de simulação foram desenvolvidos para estas áreas [2].

A plataforma *Decision Support System for Agrotechnology Transfer Cropping System model Croppin System Model* (DSSAT-CSM) é uma das mais difundidas aplicação de modelos de simulação na agricultura com o uso de modelos de crescimento e desenvolvimento com integração de sistemas de suporte à tomada de decisão [2] . Além de englobar os modelos de simulação de culturas, ele também apresenta um sistema gerenciador de base de dados e um módulo de aplicativos para avaliação de estratégias tecnológicas [3]. Como vários dos modelos mais antigos, foram desenvolvidos na linguagem Fortran, e devido as suas limitações e restrições, tornando-se de difícil manutenção e evolução.

A integração de modelos de simulação era um trabalho bastante complicado e oneroso, pois não havia padrões e estruturas de desenvolvimento, e em cada estudo criava-se uma nova estrutura. A construção e integração dos modelos dependia basicamente do conhecimento prévio do modelo e os pontos de acoplamento para a incorporação de novos módulos ou mesmo de outro modelo de simulação, tornando-se uma tarefa com complexidade muito elevada para o desenvolvedor que não tem pleno conhecimento sobre o modelo [4].

Hoje em dia, a criação de novos modelos tende a seguir algumas características como a modularização dos processos, onde os módulos do modelo são facilmente acoplados, facilitando a substituição, o reaproveitamento e a evolução de modelos. Entretanto, antigamente para cada novo estudo era desenvolvido um novo padrão, sendo necessário que o desenvolvedor realize um estudo aprofundado do modelo para realizar o acoplamento, reaproveitamento ou evoluções [2].

Muitos destes modelos foram/são implementados em Fortran, linguagem que possui limitações como, por exemplo, na não realização de acesso a bases de dados, a *WebServices*, e a memória RAM[1], em restrições na visualização dos dados e na interoperabilidade com outras linguagens de programação.

No capítulo 2 será tratado os conceitos básicos necessários para o desenvolvimento deste trabalho. No capítulo 3 será apresentado um *middleware* genérico que permita a integração de modelos de código fonte legado com tecnologias modernas e outras linguagens de programação encapsulado em um pacote R. No capítulo 4 é apresentando um *middleware* que realiza a gerência de entrada e saída dos dados necessários para a execução dos modelos presentes na plataforma DSSAT-CSM. No

[1] Memória de acesso aleatório

capítulo 5 será exposto as considerações finais deste trabalho. Por fim, no capítulo 6 será apresentada as conclusões obtidas com este trabalho.

2. REVISÃO DE LITERATURA

Neste capítulo será apresentada uma revisão com as principais características das abordagens de acoplamento de modelos de códigos legados. Será conceituado a criação de pacotes na linguagem R e seus padrões e apresentado os conceitos sobre a suíte DSSAT. Uma descrição do modelo de simulação genérico SimpleCrop será apresentada. Além disso, será apresentado uma revisão sobre métodos de acoplamento de modelos com código fonte legado encontrados na literatura.

2.1 TÉCNICAS DE ACOPLAMENTO DE MODELOS DE SIMULAÇÃO

A escolha de uma abordagem apropriada de integração para os modelos em desenvolvimento é excepcionalmente importante para desenvolver o acoplamento dos modelos, devido a grande complexidade envolvida no processo de integração [2].

As abordagens mais utilizadas para realizar a integração de modelos de simulação citadas no meio acadêmico são a monolítica, a programada, a orientada a componentes e a orientada a comunicação [4].

2.1.1 Abordagem monolítica

A abordagem monolítica é definida como sendo a criação de um programa por meio de um único código fonte, onde se encaixa fragmentos de código fonte de dois ou mais modelos, gerando assim um novo modelo customizado, como pode ser visto na Figura 1, onde retira-se trechos de código de duas fontes diferentes. [4].

Figura 1. Abordagem monolítica: trechos de dois ou mais códigos são utilizados para a criação de um novo código fonte [2].

Na utilização deste tipo de abordagem obtém-se um certo grau de facilidade para realizar a integração dos códigos dos modelos, devido ao fato de se trabalhar com apenas uma linguagem de programação. Outro fator que deve ser destacado é o controle sobre todos os detalhes do código

fonte do modelo, como, por exemplo, sua estrutura de controle, seus formatos de entrada e saída dos dados, seus tipos dos dados, sua alocação de memória dentre outros [4].

Há alguns detalhes que podem dificultar a aplicação deste tipo de abordagem, como, por exemplo, a necessidade de um completo entendimento da estrutura do modelo, dificuldade existente em se trabalhar com códigos legados e as dificuldades na aplicação de técnicas de engenharia de software (testes, verificações, atualizações etc.) [4].

Esta abordagem foi posta em prática no acoplamento de dois modelos de simulação desenvolvidos em uma mesma linguagem de programação, o CROPGRO-Tomato, que simula o crescimento e o desenvolvimento do tomate em conjunto com o modelo de doenças do tomate FOBIS-TMED. O acoplamento foi realizado por meio da inclusão dos códigos fontes do modelo FOBIS-TMED dentro do modelo CROPGRO-Tomato. Este acoplamento possibilitou contabilizar danos causados por duas doenças que influenciam no crescimento desta planta [2].

2.1.2 Abordagem programada

Na técnica da abordagem programada, apresentado na Figura 2. Os modelos permanecem como programas independentes, um não interferindo no outro em tempo de execução, significando que as saídas dos dados de um modelo integrado dentro desta abordagem são utilizadas como entrada de dados para outro modelo [4].

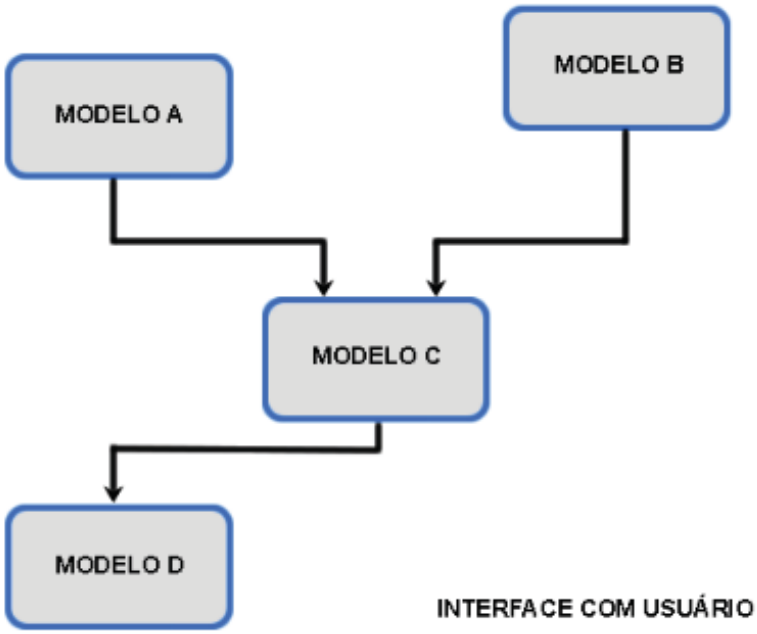

Figura 2. Abordagem programada: saídas dos modelos são conectadas como entradas para outros modelos [2].

A abordagem fornece um sistema automatizado no qual o pesquisador pode selecionar os modelos e os conjuntos de dados e a ordem de execução dos modelos de integração. Contudo, diferentes interfaces entre os modelos necessitam do desenvolvimento de padrões que possibilitam a

troca de dados por meio da interface, como a transformação de dados (mesmas unidades e medidas), que necessitam da substituição de todas as entradas e das saídas dos modelos que não sigam os mesmos padrões [4].

Como Pavan [5] explana em seu trabalho, pode-se observar o uso desta abordagem de acoplamento na utilização do modelo de simulação CROPSIM, que é o responsável por simular o crescimento e desenvolvimento do trigo em conjunto com o modelo de doença GIBSIM. A integração se dá por meio do uso das saídas do modelo CROPSIM como entrada no modelo GIBSIM (Figura 3).

Figura 3. Abordagem programada: integração entre os modelos CROPSIM e GIBSIM [2].

2.1.3 Abordagem orientada a componentes

A abordagem orientada a componentes é parecida com a abordagem monolítica, onde o resultado da integração é um único modelo. Esta abordagem decompõem os códigos dos modelos em componentes de software, no qual os componentes são sub-rotinas modulares reutilizáveis. A modularidade significa que os componentes podem ser escritos com pouco conhecimento de outros componentes do modelo e podem ser substituídos independentemente sem impactos significantes no resto do modelo. A reutilização (Figura 4) permite que um componente possa ser utilizado em uma variedade de situações sem a necessidade de realizar alterações no componente [2].

A execução dos componentes é encapsulada dentro do componente, onde o pesquisador necessita apenas conhecer os tipos de dados utilizados na entrada e na saída do componente. Com a utilização de componentes, há a possibilidade de aplicar facilmente as técnicas de engenharia de software, como, por exemplo, testes, atualizações, comparações e verificações dentre outras, sendo facilmente agregados ou reagrupados em novas construções e podem ser reutilizados em composições futuras [2].

Na utilização desta abordagem de acoplamento de modelos em códigos legados, existe a necessidade de se ter total conhecimento sobre os códigos dos modelos, além de ser necessário se ter conhecimento sobre a ordem de execução do modelo em uma possível conversão de componentes, o quê pode exigir grande reprogramação dos códigos existentes, porém, esta conversão, na grande maioria das vezes, pode se tornar inviável [2].

Um exemplo de uso desta abordagem se dá por meio de um modelo de simulação de desenvolvimento de culturas implementado em R. Este modelo é dividido em três partes principais: módulos,

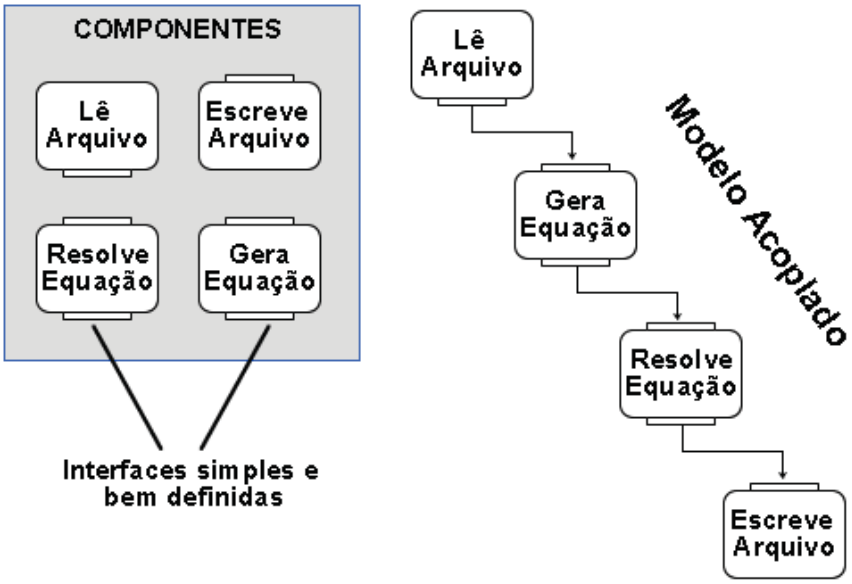

Figura 4. Abordagem orientada a componentes: componentes conectados para a criação de modelos [2].

modelos de plantas e modelos de doenças e cada parte do modelo possui diversas subdivisões descrito na Figura 5 [6].

2.1.4 Abordagem de comunicação

A abordagem de comunicação é mais complexa, porém a mais utilizada, pois permite a integração de códigos legados com tecnologias atuais aumentando, assim, a vida útil do modelo legado.

Os códigos desta abordagem permanecem independentes com a integração realizada através da troca de mensagens durante o tempo de execução, como pode ser visto na Figura 6 . As funções primárias de uma interface de troca de dados são a constituição dos fluxos e a transformação dos dados, realizando, algumas vezes, o controle da inicialização do modelo ou o estado global da integração [2].

Modelos de simulação desenvolvidos nesta abordagem de acoplamento podem ser classificados pelo uso ou não de aplicações independentes que intermediam a execução e a comunicação entre modelos. Sem a utilização da interface de acoplamento, são consideradas as bibliotecas de transferência de dados, rotinas customizadas para conversão dos dados e para a definição do estilo de comunicação utilizados pelos modelos. Com a utilização da interface de acoplamento, usam bibliotecas de comunicação que suportam diretamente a interface modelo a modelo tão bem como suportam a interface modelo à interface de acoplamento. Em ambos os casos, as bibliotecas necessitam da conversão dos dados do modelo dentro de um padrão [2].

Esta abordagem evita a necessidade da reescrita dos códigos fontes necessários pela abordagem monolítica e pela a abordagem de componentes, porém, ainda, há a necessidade de conhecer

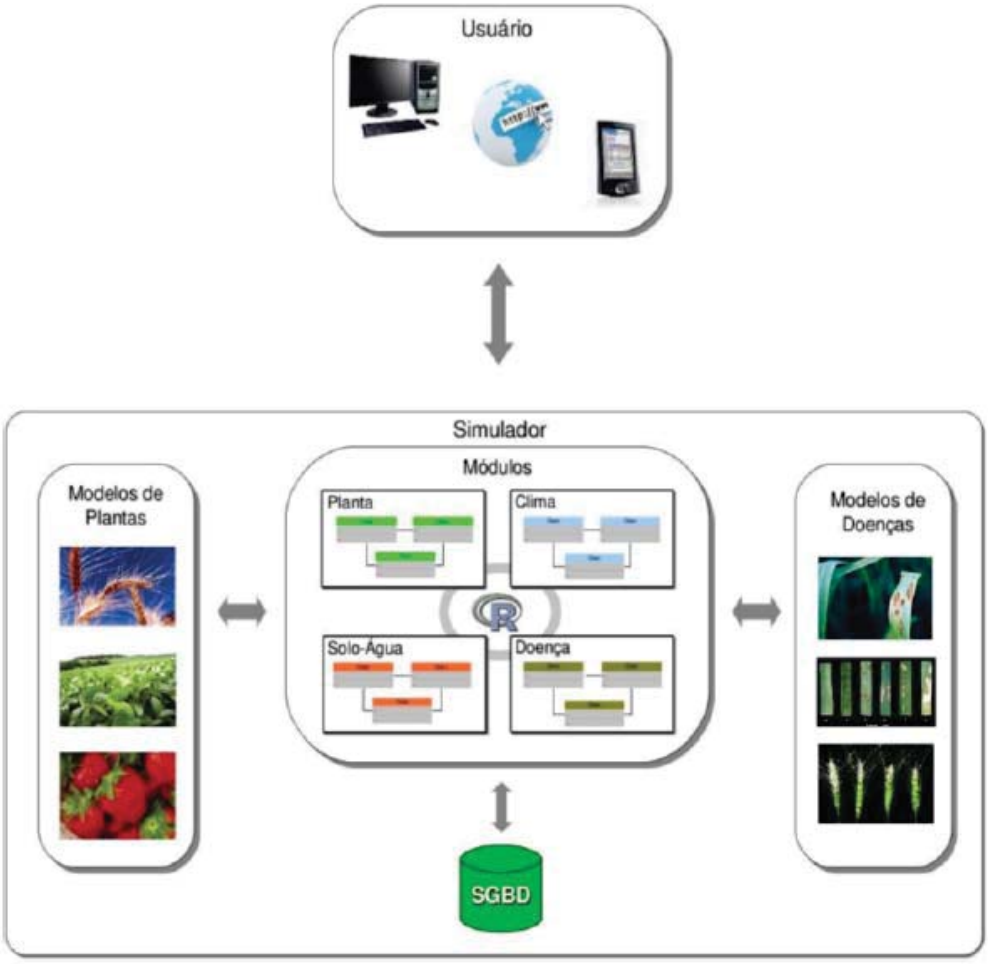

Figura 5. Abordagem orientada a componentes: exemplo desenvolvido por Rech et. al. [6].

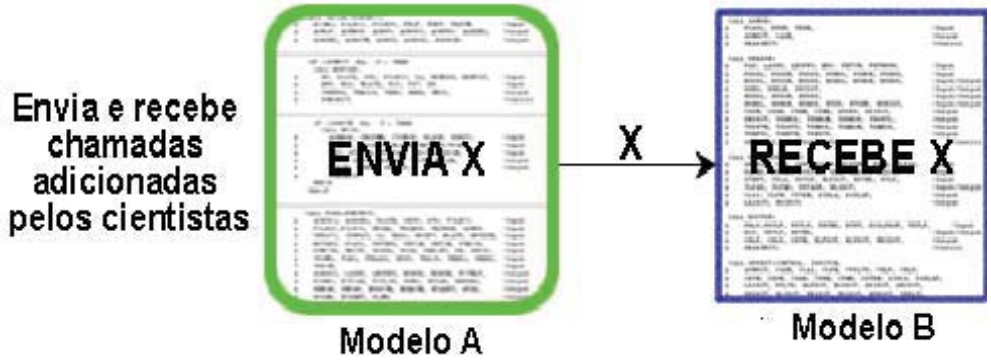

Figura 6. Abordagem de comunicação: os modelos são modificados, podendo realizar troca de dados em tempo de execução [2].

detalhadamente os códigos dos modelos para realizar o acoplamento. Inconsistências na comunicação entre os modelos podem interferir diretamente nos resultados obtidos [2].

O desempenho da execução de modelos integrados depende muito de como é desenvolvida a troca de dados. Caso a interface de comunicação seja requisitada muitas vezes durante a execução e caso existam muitos passos para a troca dos dados, pode acarretar em uma drástica desaceleração no tempo de execução deste modelo [2].

A abordagem orientada a comunicação prova ser o melhor caminho a seguir na construção de novos modelos se os componentes já estiverem disponíveis para a integração, caso contrário, esta abordagem não é a mais ideal para acoplar os modelos existentes. Ela também permite que modelos existentes sejam acoplados com mínimas mudanças nos códigos fontes [4].

Um exemplo desta abordagem foi aplicada com um modelo genérico de doenças e integrado com o modelo CROPGRO-Soybean que simula o crescimento e desenvolvimento da soja, tendo como objetivo contabilizar o impacto e o dano que a doença da ferrugem asiática causa na soja. A interface (Figura 7) desenvolvida utiliza JNI para a comunicação entre a linguagem Java do modelo genérico de doenças com a linguagem Fortran do modelo CROPGRO-Soybean [2].

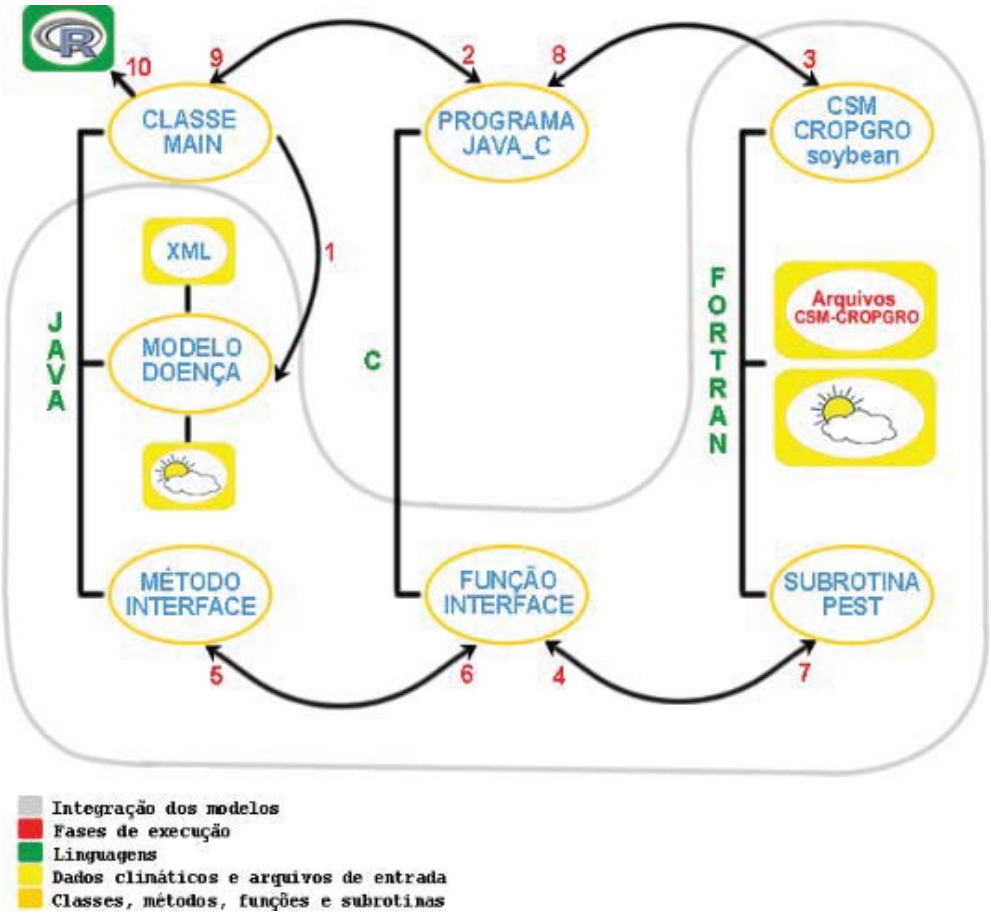

Figura 7. Abordagem de comunicação: integração do modelo genérico em Java com o modelo CROPGRO-Soybean em Fortran [2].

2.2 CRIAÇÃO DE PACOTES EM R

O sistema de pacotes tem sido um dos fatores chaves do grande sucesso global do R, pois permite manter, de uma maneira fácil, coleções de funções, conjuntos de dados e documentação, podendo ser distribuído de uma maneira simplificada. Os pacotes em R permitem ser carregados e descarregados em tempo de execução e, assim, ocupando memória apenas quando são utilizados [7].

A criação de pacotes em R é feita com a utilização de recursos oferecidos pelo próprio R e, independentemente da plataforma utilizada, as formas de criação são as mesmas. No entanto, as con-Figurações de armazenamento dos sistemas operacionais são diferentes, dependendo da plataforma computacional. Deste modo, os pacotes no R seguem um padrão de diretórios interno, definido pela sua equipe de desenvolvedores, com a finalidade de padronizar e deixar os pacotes com a mesma característica, fazendo com que os erros ocasionados pelo pacote sejam vinculados unicamente a ele mesmo e não ao R [8].

2.3 DECISION SUPPORT SYSTEM FOR AGROTECHNOLOGY TRANSFER (DSSAT)

O DSSAT é um sistema de apoio à decisão para a agrotecnologia composto por modelos de simulação de culturas, disponível para mais de 42 culturas diferentes. Disponibiliza um conjunto de ferramentas que visam facilitar a avaliação e a aplicação dos modelos [3]. Inclui aplicações que auxiliam na análise sazonal e sequencial dos riscos econômicos e impactos ambientais ocasionados pela irrigação, pelo uso de fertilizantes e de nutrientes, pelas mudanças climáticas, pela variabilidade climática e pela agricultura de precisão [9].

Desenvolvido pelo projeto IBSNAT (*International Benchmark Sites Network for Agrotechnology Transfer*) em conjunto com a Universidade do Havaí em meados dos anos 1974, a fim de facilitar a aplicação de modelos de culturas em uma abordagem sistêmica à pesquisa agronômica [9]. Mantido atualmente pela *DSSAT Foundation*[2] entidade que reúne um grande número de pesquisadores de diversas universidades e centros de pesquisa.

Atualmente o DSSAT é uma das ferramentas mais conhecidas e difundidas na agrotecnologia sendo aplicada nas mais diversas áreas como, no manejo agrícola, na precisão e nos impactos da alteração climática. E vem sendo usado por mais de 40 anos por pesquisadores, educadores, consultores em mais de 100 países [3].

O DSSAT é uma coleção de programas independentes que operam em conjunto, modelos de simulação de culturas estão no centro (núcleo da suíte), como apresentado na Figura 8. As bases de dados descrevem o clima, o solo, as condições e as medições do experimento e informações genotípicas para a aplicação dos modelos em diferentes situações. O software auxilia o usuário no preparo das bases de dados e na comparação dos resultados simulados com os observados para dar validade nos modelos ou determinar se são necessárias modificações para melhorar a precisão [9].

O DSSAT realiza a simulação do crescimento, do desenvolvimento e do rendimento de uma cultura que cresce em uma área uniforme de terra sobre a administração prescrita ou simulada, bem como as mudanças de água no solo, carbono e nitrogênio que o ocorrem no sistema de cultivo ao longo do tempo [9].

O DSSAT é dividido em um programa principal e um módulo de unidade de terra, que possuí módulos para componentes primários em um sistema de cultivo (Figura 9). Os módulos primários são destinados para o clima, a planta, o solo, interface entre solo-planta-atmosfera e os componentes de

[2] http://dssat.net/

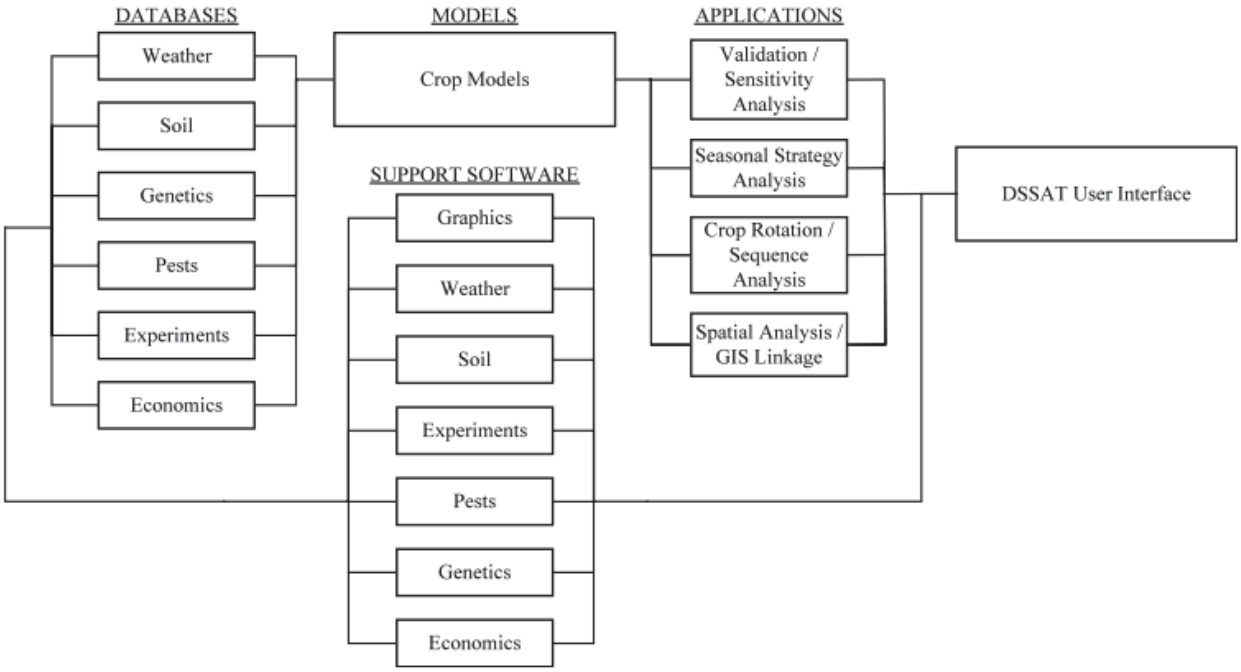

Figura 8. Diagrama de integração entre bases de dados, aplicação e componentes de suporte de Software e seu uso em conjunto com os modelos de cultura em aplicação no DSSAT [9].

gerenciamento. Coletivamente, esses componentes descrevem as mudanças do tempo no solo e nas plantas que ocorrem em uma única unidade de terra em resposta ao clima e o manejo [9].

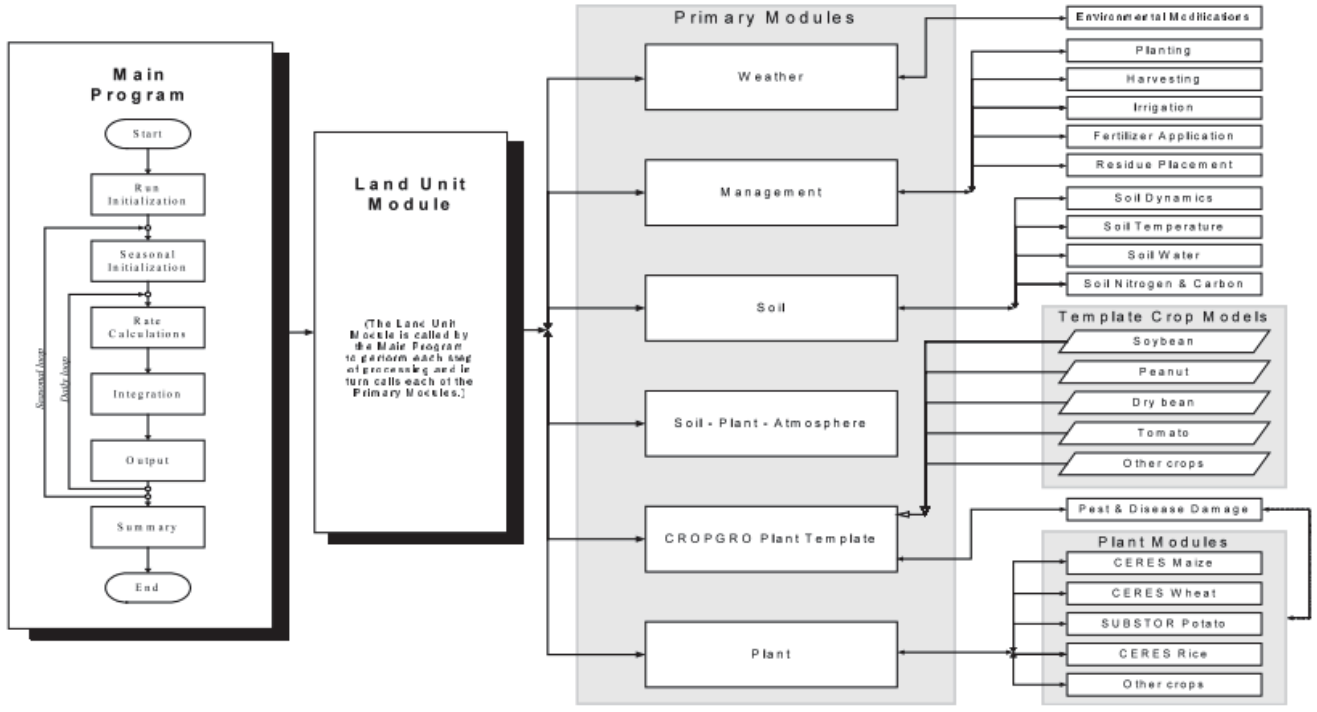

Figura 9. Visão geral da estrutura modular do DSSAT [9].

Cada módulo tem seis estados operacionais (Figura 9), os quais são: inicialização da execução, inicialização da temporada, cálculos de taxa, integração, saída diária e saída do sumário. O

módulo principal é responsável por controlar quando cada um desses estados está ativo, e quando cada módulo deve executar a tarefa. Os módulos podem ser divididos entre módulos primários e sub módulos; Cada sub módulo normalmente executa cada um dos seis estados operacionais [9].

O programa principal lê os dados dos arquivos padrões DSSAT, os quais descrevem um experimento ou uma situação a ser simulada, e define várias variáveis para o controle da execução da simulação. Quando a simulação é iniciada o módulo de unidade de terra é chamado para definir as variáveis que devem ser definidas no inicio de cada temporada. Logo após a inicialização do *loop* sazonal, o programa principal inicia um *loop* diário chamando o módulo de unidade terra três vezes seguidas, a primeira para calcular as taxas, segundo para integrar as variáveis de estado e a terceira realizar as saídas diárias. Depois que a temporada de colheita é terminada o módulo de unidade de terra é chamado para gerar os arquivos de saída [9].

O DSSAT tem 7 módulos principais, os quais são: clima, solo, solo-planta-atmosfera,CROPGRO template, planta, manejo e módulo de peste, como pode ser visto na Figura 9.

2.3.1 Módulo de clima (Weather)

O módulo de clima, como pode ser visto na Figura 9, tem como principal função ler ou gerar os dados climáticos diários: temperatura máxima e mínimas do ar, radiação solar, precipitação, umidade relativa e velocidade do vento, quando disponíveis do arquivo climático diário. Os valores climáticos são calculados de hora em hora quando algum módulo necessitar [9]. Os dados climáticos diários são gerados por meio de dois geradores meteorológicos: WGEN (*Weather Generation Model* [10, 11] ou SIMMETEO [12, 13].

O módulo também pode modificar as variáveis climáticas para serem realizados estudos das mudanças climáticas ou experimentos onde a radiação solar, temperaturas máximas ou minimas, o comprimento do dia, os valores de CO2 contidos na atmosfera, podem ser valores constantes ou diminuir/aumentar em relação aos dados de entrada. Com base na conFiguração da simulação o módulo é capaz de identificar se ele deve ler, gerar ou modificar (usando o sub módulo de modificações ambientais Figura 9) os dados climáticos diários [9].

2.3.2 Módulo de solo (Soil)

O solo é representado como um perfil unidimensional, homogêneo horizontalmente com um número de camadas verticais [9], sendo composto por 4 sub módulos, os quais são: água no solo, temperatura do solo, carbono e nitrogênio no solo e dinâmica do solo.

2.3.2.1 Sub módulo de água no solo (Soil water)

O sub módulo de balanço hídrico do solo foi desenvolvido primeiramente para o modelo CERES-Wheat [14], e foi adaptado para ser utilizado por todos os modelos do DSSAT a partir da versão 3.5 [15, 16, 17]. Sendo um modelo unidimensional que calcula as alterações diárias da quantidade de água contida nas camadas do solo devido a infiltração da precipitação e irrigação, drenagem vertical, fluxo não saturado, evaporação, transpiração da planta e captação da água foram separados em um módulo solo-planta-atmosfera (SPAM) [9].

O módulo apresenta uma abordagem "*tipping bucket*", a drenagem da água no solo é calculada quando o teor de água de uma camada inferior atinge um valor a cima do valor máximo de drenagem de uma camada superior. O fluxo insaturado ascendente também é calculado usando uma estimativa conservadora da difusividade de água no solo e de diferenças no teor volumétrico de água no solo das camadas adjacentes [17].

A infiltração de água no solo que ocorre durante o dia é calculada subtraindo-se o escoamento superficial da precipitação ocorrido durante o dia [9]. A precipitação é dividida em escoamento e infiltração através do metódo SCS (*Soil Conservations Service*) [18], com base em um número de curva que tenta explicar a textura, declive e preparo do solo. O método foi alterado por Williams et. al. [19], e eles explicam os solos em camadas e o teor de água no solo no momento que ocorre a precipitação.

Quando a irrigação é aplicada, a quantidade de água é adicionada ao montante de precipitação diária para ser realizado o cálculo de infiltração e escoamento. A drenagem da água através do perfil é feita primeiramente com base num parâmetro global de drenagem do solo assumindo constante com a mesma profundidade. A quantidade de água que passa através de qualquer camada é, então, comparada com a condutividade hidráulica saturada dessa camada. Se a condutividade hidráulica saturada de qualquer camada é inferior ao valor de drenagem vertical calculada, a drenagem real é limitada ao valor da condutividade e a água se acumula acima da camada [9].

A evaporação da água na superfície do solo e a captação da água pela raiz (transpiração) de cada camada são calculadas pelo módulo de solo-planta-atmosfera (SPAM) e devolvido para o sub módulo de água no solo. Cada dia o teor de água no solo de camada é atualizado adicionando ou subtraindo fluxos diários de água para a camada [9].

2.3.2.2 Sub módulo de balanço de carbono e nitrogênio do solo (Soil carbon and nitrogen balance)

O DSSAT tem duas opções para simular a matéria orgânica do solo e o balanço de nitrogênio [9]. O modelo original de matéria orgânica do solo do DSSAT [20, 21] foi baseado no modelo PAPRAN de Seligman e Van Keulen [22]. O outro módulo de matéria orgânica do solo foi desenvolvido por Gijsman et al. [23] baseado no modelo CENTURY [24, 25].

O modelo baseado no CENTURY distingue em três tipos a matéria orgânica no solo: fácil decomposição (microbiana) tipo 1, recalcitrante tipo 2, o qual contêm Lignina e paredes celulares, e

um quase inerte tipo 3. Na inicialização da simulação, a proporção fracionária dos três conjuntos é definido, o tipo 1 com aproximadamente 2% do total de matéria orgânica no solo, enquanto o tipo 2 e o tipo 3 variam conforme o histórico de manejo do solo (pastagens ou cultivadas) e o grau de esgotamento [9].

2.3.2.3 Sub módulo de temperatura do solo (Soil temperature)

O modelo de simulação da temperatura do solo presente no DSSAT foi originalmente derivado do modelo EPIC [19]. A temperatura do solo é calculada a partir da temperatura do ar e a condição limite de temperatura profunda do solo é calculada através da média da temperatura anual do ar e a amplitude das médias das temperaturas mensais [9].

Também disponibiliza uma abordagem simples para calcular o impacto da radiação solar e coeficiente de reflexão na temperatura da superfície do solo. A temperatura do solo é usada para modificar os processos da planta e a decomposição da matéria orgânica do solo [9].

2.3.3 Módulo de solo-planta-atmosfera (Soil-plant-atmosphere)

Este módulo calcula a evaporação do solo e a transpiração da planta diária, e foi desenvolvido por Ritchie [26] e integrado no DSSAT como parte do balanço hídrico do solo, ele recebe como entrada os dados do solo, da planta e da atmosfera e realiza os cálculos de interceptação de luz pela copa, evapotranspiração potencial . Também é responsável pelo cálculo da quantidade de água retirada pela raiz de cada camada do solo [9].

São necessários como entrada os dados diários do clima, todas as propriedades do solo e a água contida por camada, o índice da área foliar e o comprimento e a densidade da raiz para cada camada [9].

O método Penman-FAO [27] pode ser utilizado opcionalmente para se obter melhores resultados para condições áridas e com vento, mas os dados climáticos devem conter dados de vento e umidade.

A evapotranspiração potencial é particionada em: potencial evaporação do solo baseada na fração da energia solar que atinge a superfície do solo, com base em uma função exponencial negativa do índice da área foliar e potencial transpiração da planta [9]. Após a superfície ser molhada pela primeira vez devido a precipitação ou irrigação, a evaporação ocorre na taxa potencial até atingir a quantidade acumulada de evaporação do solo desde que o molhamento seja atingido [9]. Em seguida, a quantidade limite de evaporação diária do solo é calculada como uma função de raiz quadrada do tempo desde que a fase tenha terminado. Se a evaporação é menor que o potencial de evaporação do solo, a diferença é adicionada ao potencial de transpiração da planta para se ter em conta o aumento da carga de calor na copa quando a superfície do solo estiver seca [9].

Para determinar se o solo ou a atmosfera limitam a transpiração da planta, a potencial captação de água pela raiz é calculada através do cálculo de um fluxo máximo de água para as raízes em cada camada e somando esses valores [14, 17, 15].

A equação que calcula a captação potencial de água pela raiz em cada camada é uma aproximação à equação de fluxo radial, onde são feitas suposições sobre o efeito da textura do solo na condutividade hidráulica, diâmetro da raiz e uma diferença máxima de potencial de água entre as raízes e o solo. A transpiração real da planta é então calculada como o mínimo de transpiração potencial da planta e a potencial absorção de água pela raiz. Assim, a atmosfera pode limitar a transpiração por baixa radiação solar e baixas temperaturas, a copa pode limitar por um baixo índice de área foliar e o solo pode limitar pelo baixo teor de água no solo, baixa densidade das raízes [9].

2.3.4 Template CROPGRO

O módulo de template CROPGRO é o mesmo descrito por Boote et al. [28]. Foi desenvolvido como uma abordagem genérica para modelar culturas com um código fonte em comum, podendo prever uma série diferente de culturas [9].

O módulo simula dez culturas, incluindo sete leguminosas de grão: soja, amendoim, feijão seco, grão-de-bico, feijão Caupi, feijão de veludo, fava de feijão e também de não leguminosas como tomate [9].

As características de cultivar e de ecótipo são definidos nos arquivos entrada, e o módulo é responsável por realizar as suas leituras. As cultivares são definidas por 15 características diferentes; duas características de sensibilidade ao comprimento do dia, cinco importantes da duração do ciclo de vida, fotossíntese foliar da luz saturada, características vegetativas e características reprodutivas. Também há 19 características no arquivo de ecótipo que foram propostos para variarem com menos frequência, como o tempo térmico até a emergência e os primeiros estádios da folha [9].

A fenologia é um importante componente, porque usa dados do arquivo de especies, que contém os valores cardeais, bem como dados dos arquivos de cultivares e ecótipos, os quais contêm durações fisiológicas do dia para as respectivas fases do ciclo de vida. O progresso do ciclo de vida através de uma determinada fase depende de um acumulador fisiológico do dia em função da temperatura e do comprimento do dia. Algumas culturas como a da soja são sensíveis ao comprimento do dia, já outras, não, como a do amendoim. Quando o acumulador de dias fisiológicos atinge o valor limiar definido pelo arquivo de cultivares, um novo estágio de crescimento é desencadeado [9].

O arquivo de espécies também contém os coeficientes que indicam o efeito do déficit de água ou nitrogênio na taxa de progresso do ciclo de vida. Estes coeficientes podem variar conforme a fase do ciclo de vida. Por exemplo, o déficit hídrico pode retardar o início do crescimento produtivo, mas acelera o crescimento reprodutivo após o início do preenchimento da semente [9].

2.3.5 Módulo de planta (Plant)

O módulo de planta serve como interface para modelos de culturas individuais acessarem os outros módulos do DSSAT, projetado para ligar os módulos que descrevem o crescimento, desenvolvimento e rendimento com modelos de culturas individuais como, por exemplo, os modelos CERES que, após algumas alterações, estão aptos a acessar todos os módulos que compõem o DSSAT [9].

O DSSAT tem diversos modelos de culturas individuais, como milho, trigo, sorgo, painço, cevada, arroz e batata, e outros que podem ser facilmente adicionados [9].

O crescimento das culturas são calculados como, por exemplo, nos modelos CERES: o ciclo de vida da planta é divido em várias fases, as quais são semelhantes entre as culturas, sendo a taxa de desenvolvimento controlada pelo tempo térmico ou pelos Graus-dias (GDD) [9]. Graus-dias é o método que baseia-se na premissa de que uma planta necessita uma certa quantidade de energia, representada pela soma de graus térmicos (médias das temperaturas máximas e minimas diárias) acima de uma temperatura base para se desenvolver [29]. Os Graus-dias são necessários para ser realizado a progressão de um estadio de crescimento para outro, são de definidos pelo usuário ou são calculados internamente com bases nós dados de entradas e em pressuposições sobre a duração dos estádios intermediários [9].

O comprimento do dia pode afetar no número total de folhas formadas pela alteração da duração da fase de indução floral e, portanto, a iniciação floral. Se a execução será sensível ao comprimento é uma entrada específica do arquivo de cultivar [9].

Durante a fase vegetativa, a emergência de novas folhas é usada para o limitar o desenvolvimento da área foliar até que o número dependente da espécie de seja atingida. Posteriormente, pode ocorrer a ramificação vegetativa, e o desenvolvimento da área foliar depende da disponibilidade de assimilados e área foliar específica. A sua expansão é modificada pelo GDD, pela água e o estresse de nitrogênio [9].

O crescimento diário da planta é calculado convertendo a radiação ativa fotossinteticamente interceptada diariamente pela matéria seca da planta. A interceptação da luz é calculada como uma função de índice de área foliar, população de plantas e espaçamento entre fileiras. A quantidade de matéria nova seca disponível para o crescimento, a cada dia, também é suscetível aos efeitos de água limitada ou estresse de nitrogênio, temperatura e é sensível a concentrações de CO2 na atmosfera [9].

A biomassa que está acima do solo tem prioridade para os carboidratos e, no final do dia, os carboidratos não utilizados são alocados para as raízes. As raízes devem receber um mínimo dependente de um estágio especificado de carboidratos diários disponível para o crescimento. A área foliar é convertida em peso de folha nova, usando funções empíricas [9].

A quantidade de núcleos por planta é calculada durante a floração com base no potencial genético da cultivar, peso da copa, taxa média de acumulação de carboidratos, temperatura, água e estresse de nitrogênio. A quantidade potencial de núcleos é uma entrada definida pelo usuário para

cada cultivar especifico. Uma vez atingido o início do enchimento do grão, o modelo calcula a taxa de crescimento diário com base em valores especificados pelo usuário [9].

2.3.6 Módulo de manejo (Management)

O módulo de manejo determina quando foram executadas as operações no campo através dos sub módulos. As operações realizadas são: plantio, colheita, aplicação de fertilizantes inorgânicos, irrigação e aplicação de resíduos de colheita e material orgânico [9]. Estas operações podem definidas pelo usuário no arquivo de experimento padrão [30].

As operações são definidas pelos usuários se devem ser automáticas ou fixas com base nas datas de entrada ou dias após o plantio. As condições que causam o plantio automático dentro do intervalo de tempo são o teor médio de água no solo e uma profundidade especificada e a temperatura do solo a uma profundidade especificada entre limites especificados [9].

A colheita pode ocorrer em algumas situações como em determinadas datas, quando a cultura estiver madura ou quando as condições da água no solo no campo são favoráveis para o funcionamento da máquina [9].

A irrigação pode ser aplicada em datas específicas com uma quantidade de irrigação especificada ou pode ser controlada pela quantidade de água disponível na planta. Caso a quantidade de água disponível na planta cair abaixo de uma determinada fração da capacidade de retenção em uma profundidade de manejo de irrigação, um evento de irrigação é acionado. A quantidade de irrigação aplicada pode ser uma quantidade fixa ou pode preencher o perfil até a profundidade de manejo [9].

Os fertilizantes podem ser aplicados em datas fixas com quantidades especificadas ou as aplicações podem ser opcionalmente controladas quando a planta necessitar de nitrogênio através do módulo de planta [9]. Os resíduos de cultura e os adubos orgânicos são aplicados no início da simulação, após a colheita ou em datas fixas semelhantes as dos fertilizantes inorgânicos [9].

Estas opções de manejo permitem aos usuários uma grande flexibilidade para simular experimentos que foram realizados no passar para avaliar melhorias nos modelos e para simular sistemas de manejo opcionais para diferentes aplicações [9].

O arquivo de gerenciamento também fornece um escopo que permite definir várias culturas e estrategias de manejo para as rotações de culturas e sequenciamento [9].

2.3.7 Módulo de peste (Pest)

O módulo de peste foi desenvolvido para os modelos CROPGRO [31]. Permite ao usuário inserir observações do campo, dados de reconhecimento da população de insetos, gravidade da doença em diferentes tecidos vegetais ou danos físicos nas plantas ou em seus componentes, afim de simular os efeitos de pragas e doenças no crescimento e rendimento.

Os resultados dos processos de crescimentos das plantas se da através da redução da área foliar, da perda de assimilação, de perda de folhas, dos frutos, dos caules ou das raízes e da inativação da capacidade fotossintética das folhas [9].

2.4 SIMPLE CROP

O modelo Simple Crop foi desenvolvido com o objetivo de demonstrar a abordagem para a construção de modelos com estrutura modular proposta por Reynolds e Acock [32] e Acock e Reynolds [33], e consiste em quatro partes principais: o programa principal, módulo de crescimento da planta, módulo de balanço hídrico do solo e módulo de clima, desenvolvido inteiramente em Fortran [34].

Na Figura 10 é demonstrado o formato modular do modelo desenvolvido por Porter et al. [34], onde cada módulo tem dois ou mais dos cinco componentes descritos abaixo:

- *Initialization*: realiza a leitura dos dados e a incialização das variáveis, chamado uma vez por execução.
- *Rate calculation*: realiza os cálculos das taxas de processo e das taxas de variação das variáveis de estado com base nas condições do final do dia anterior da simulação. Chamado uma vez por etapa de tempo da simulação.
- *Integration*: realiza as atualizações das variáveis de estado previamente calculadas.
- *Output*: é chamado uma vez por dia para gerar os relatórios de saída.
- *Close*: é chamado uma vez ao final da simulação para fechar os arquivos de saída e gerar os relatórios de resumo.

O programa principal contém até cinco chamadas para cada módulo para executar os vários componentes de processamento. O fluxo dinâmico de processamento dentro do programa é regulado através da variável `DYN`. No inicio da simulação cada módulo é chamado com a variável `DYN` definida como *INITIAL* que realiza a inicialização do módulo. Durante o loop de tempo diário, cada módulo é chamado três vezes pela função principal: cálculo de taxa, cálculos de integração, saída de dados diários e uma chamada final para escrever nos arquivos de saída e de resumo e após fechar os arquivos quando a simulação for concluída [34].

Os módulos de planta, de clima e de água no solo do modelo SimpleCrop, apresentados na Figura 10, serão descritos a seguir.

2.4.1 Módulo de planta

O módulo de planta realiza o cálculo do crescimento e o desenvolvimento das culturas com base nos valores diários de temperaturas máximas e mínimas, radiação solar e valor diário de dois

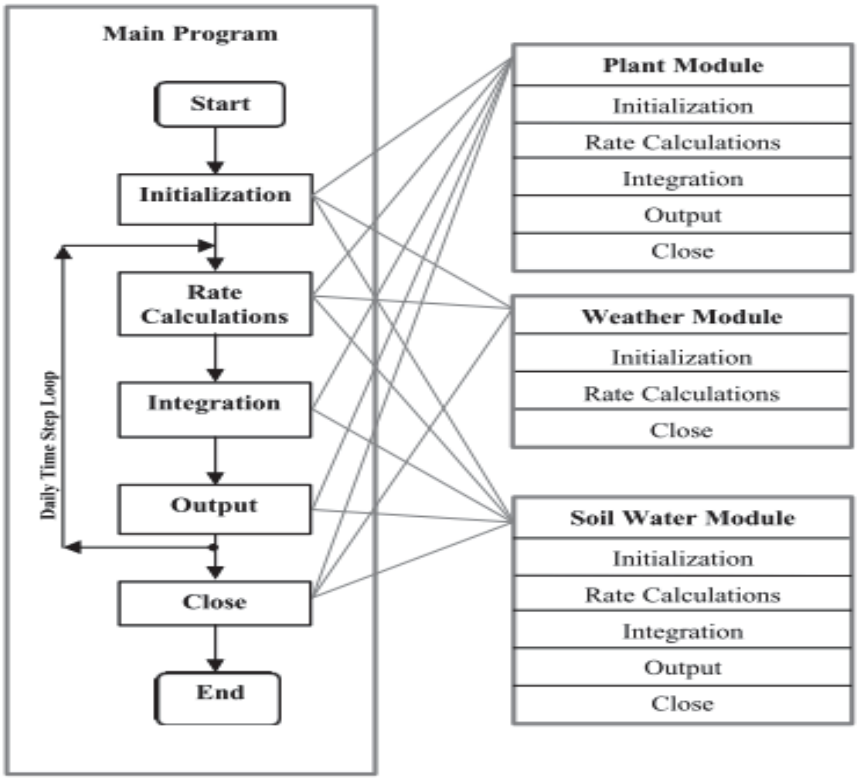

Figura 10. Estrutura modular do modelo SimpleCrop descrito por Porter et. al. [34].

fatores de estresse hídrico do solo. Também simula o índice de área foliar que é utilizado no módulo de água no solo para calcular a evapotranspiração [34].

O módulo de planta chama três sub rotinas: **PTS**, que calcula o efeito da temperatura na taxa de crescimento diário da planta e a taxa de aumento do número de folhas; **PGS**, para calcular o aumento diário do peso da planta; e **LAIS**, para calcular o aumento do índice de área foliar [34].

Na sub rotina **PTS**, a taxa de redução do crescimento é calculada todos os dias. A sub rotina **PGS**, calcula o potencial de aumento de matéria seca total diário. O ciclo da planta é divida em fases vegetativa e reprodutiva. A fase vegetativa continua até que o número máximo de folhas definidos geneticamente seja atingido. Durante a fase reprodutiva, o aumento do número de folhas é calculado com base em uma taxa máxima e um fator limitante baseado na temperatura. Durante a fase reprodutiva a diferença entre a média da temperatura diária e a temperatura base é utilizada para calcular a taxa de desenvolvimento da planta [34].

A sub rotina **LAIS** é chamada para ambas as fases (vegetativa e reprodutiva) para calcular a alteração no índice de área foliar. Durante o período vegetativo o índice de área foliar aumenta em função da taxa de aumento do número de folhas. A taxa potencial é limitada pelo estresse hídrico do solo (déficit e saturação). Depois que a planta atinge o número máximo de folhas, o índice de área foliar começa a diminuir através de uma integral térmica diária [34].

2.4.2 Módulo de balanço hídrico do solo

O solo é descrito como uma única camada homogênea subjacente por uma camada relativamente impermeável. Um balanço hídrico simples é usado para atualizar o teor de água no solo diariamente, usando os valores calculados de infiltração, evaporação, transpiração e drenagem [34].

As características do solo são: teor de água do solo no ponto de flutuação, capacidade de campo e saturação todas em unidades de cm^3/cm^3, profundidade do perfil do solo, fração de drenagem diária, número da curva e teor inicial de água no solo [34].

O módulo de balanço hídrico retorna dois parâmetros que expressam os efeitos da seca e do excesso de água no solo na taxa de crescimento da planta [34].

2.4.3 Módulo de clima

O módulo de clima é chamado três vezes pela função principal do modelo durante a execução, para a inicialização, para os cálculos de taxa e para realizar o fechamento dos arquivos.Após a inicialização, o arquivo de clima é aberto. Na seção de cálculos de taxa, os dados climáticos são lidos diariamente para o modelo[34].

2.5 RFINTEGRA

O pacote RFintegra proposto por Hölbig et al. [35], consiste basicamente em criar um pacote para a linguagem R que encapsula modelos de simulação desenvolvidos em Fortran. Definindo funções responsáveis por ler arquivos no formato tabular ASCII, e realizar a sua conversão para variáveis em R, pois a maioria dos dados atualmente ainda é fornecida neste formato de arquivo, sendo necessário o desenvolvimento de funções que realizem a conversão de variáveis R em arquivo tabular ASCII para a simulação Fortran.

Após a implementação das funções descritas anteriormente, o usuário não tem contato algum com o modelo em Fortran, tendo contato apenas com uma interface R, pois a simulação acaba se tornando uma sub rotina do R [35].

Para ter uma visão mais clara das limitações da técnica de encapsulamento de códigos Fortran no R Hölbig et. al. [35] utilizaram o modelo Simple Crop [34], já descrito anteriormente, por se tratar de um modelo simples.

O pacote RFintegra realiza três tarefas principais: converter os arquivos de texto existentes para objetos R. Realizar a conversão de objetos R para arquivos de texto para que possam ser lidos pelo modelo; e realizar a leitura e conversão dos arquivos de texto criados pelo modelo genérico de culturas em Fortran [35].

Para cada arquivo necessário pelo modelo foram desenvolvidas funções que realizam as leituras dos mesmos individualmente, tendo apenas como parâmetro o caminho do diretório que se

encontra os arquivos que ele irá utilizar na execução, se omitido este parâmetro o pacote irá utilizar os arquivos padrões distribuídos junto com o pacote [35], como pode se observar na Figura 11.

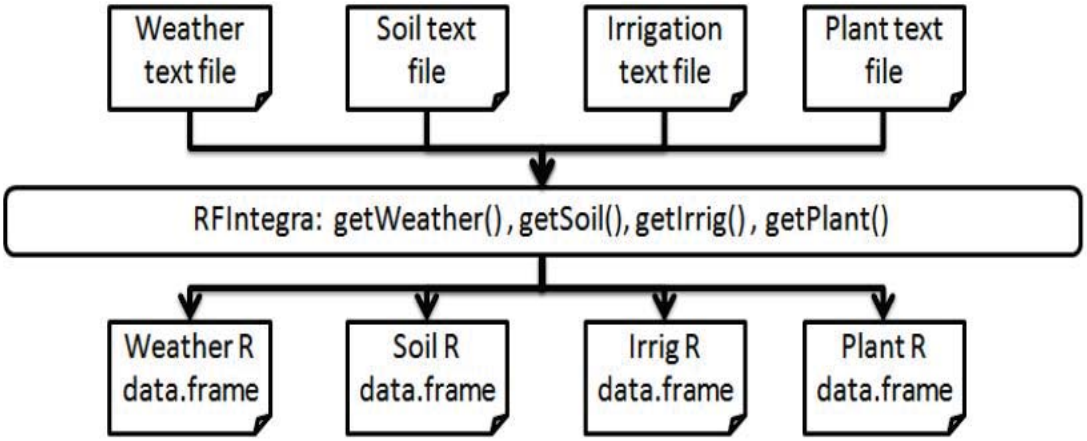

Figura 11. Estrutura de entrada dos arquivos do pacote RFintegra[35].

Os comandos de leituras (Figura 11), como por exemplo o de clima, são responsáveis por realizar a leitura dos dados do arquivo clima e convertê-los em objeto R. Assim que os dados estão disponíveis no R, é possível realizar a execução do modelo, necessitando de um `Data Frame` para cada arquivo de entrada de parâmetros necessários para o modelo de simulação [35].

Quando o modelo é executado, o pacote realiza a leitura dos `Data Frames` fornecidos e os converte em arquivos necessários para a execução do modelo em formato ASCII e os grava em disco. Após, o modelo é executado de forma tradicional pelo pacote. No término da execução, os resultados gerados pelo modelo de simulação são escritos no disco em formato ASCII. O pacote disponibiliza funções que realizam a leitura e a conversão destes arquivos em `Data Frames`, Figura 12, podendo ser manipulados como qualquer outro objeto em R [35].

A abordagem proposta por Hölbig et al. [35] teve grande sucesso em "esconder"toda a complexidade atrelada a execução do modelo de simulação desenvolvido em Fortran com código legado, sem ser necessário a alteração do código original do modelo e não precisando nenhuma validação.

2.6 APLICAÇÕES REAIS DE ACOPLAMENTOS DE MODELOS

Os modelos de simulação vem sendo cada vez mais aplicados em diversas áreas do conhecimento, como, por exemplo, Química, Matemática e Física, visando a redução de tempo e de custos da experimentação em campo. Grande parte dos modelos atendem às necessidades na forma em que foram concebidos, desenvolvidos em linguagens de programação antigas, muitas vezes sendo necessário a integração com outros modelos. A seguir são apresentados alguns trabalhos que tratam sobre a integração e o acoplamento de modelos de simulação.

Toebe [36] explana em seu trabalho uma abordagem de acoplamento de modelos heterogêneos quanto às suas tecnologias e sem necessidade de profundas alterações nas suas implementa-

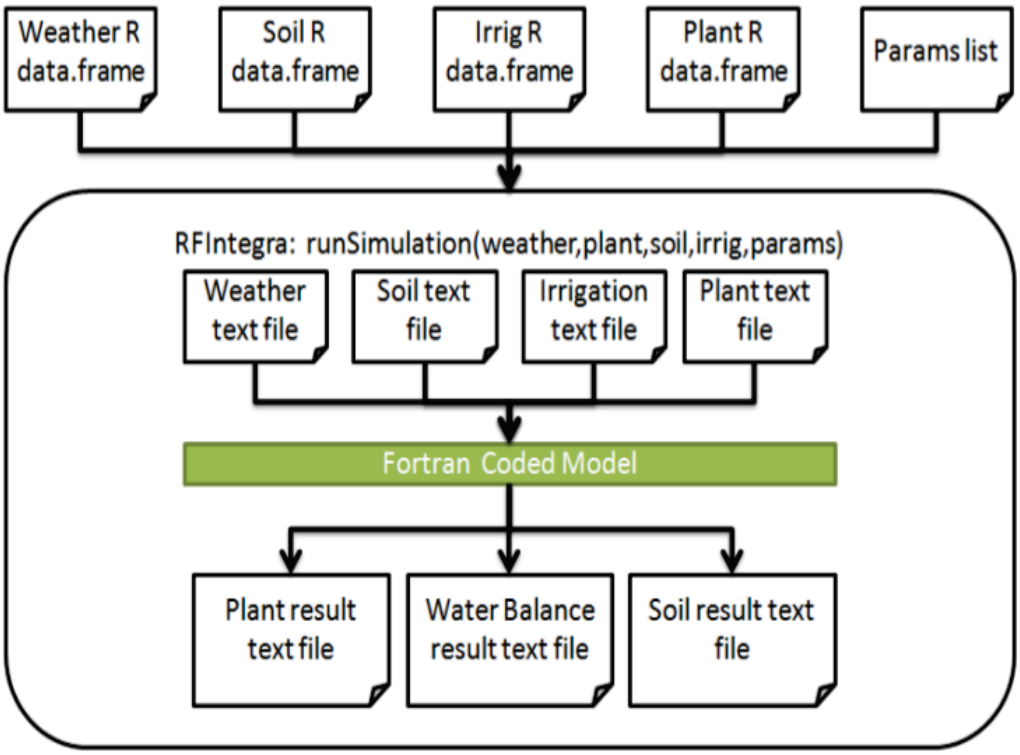

Figura 12. Interação dos objetos R com o modelo de simulação desenvolvido em Fortran [35].

ções, utilizando um sistema gerenciador de banco de dados (SGBD) como intermediário das comunicações entre os modelos e a implementação de um componente de integração.

Bergez et al.[37] apresentam uma plataforma de código fonte aberto para criar, avaliar e integrar modelos de culturas. A plataforma segue uma abordagem multidisciplinar (por exemplo, agronomia, ciência do solo, epidemiologia, ciências de gestão, estatística, matemática e ciência da computação aplicada). Permite a integração de modelos por meio da reescrita do código fonte ou pelo encapsulamento do código existente.

Brown [38] descreve um modelo de software que permite desenvolver e executar modelos de culturas na plataforma APSIM tendo como sua principal finalidade a simulação dinâmica de longo prazo em simulações agrícolas.

Em seu trabalho Wu et al. [39] explana sobre um modelo em R de acoplamento universal para facilitar o uso de funções em R para calibrar e analisar modelos de simulação. Realiza a conversão de modelos desenvolvidos em qualquer linguagem de programação em funções R utilizando o conceito de *Parameter Estimation program* (PEST) e sua estrutura operacional segue o modelo R-*Soil and Water Assessment Tool* (R-SWAT).

Ewert et al. [40] descrevem um método que permite o desenvolvimento e a utilização de modelos de simulação em forma de componentes. O modelo MODCOM fornece um conjunto de especificações de interface que descrevem os componentes de uma simulação podendo ser facilmente integrado com outros modelos.

Jansson et al. [41] elucida um software que possibilite estender facilmente códigos fontes legados desenvolvidos em Fortran. A orientação a objetos permite incluir novos módulos e interface sem alterar o código Fortran, fazendo uso de ferramentas de desenvolvimento e uma técnica desenvolvida pelos autores para o compartilhamento de memória. Pode combinar um grande número de sub-modelos, tendo seus dados de entrada e saída adaptados aos dados orientados a objetos padrão.

Jones et al. [42] apresentam uma abordagem para o desenvolvimento de modelos de forma modular, para permitir a fácil integração com o DSSAT, sendo necessário minimas alterações no código fonte do modelo original.

Nesta seção foram apresentados os principais conceitos necessários para o desenvolvimento deste trabalho. Abordagens para acoplamento de modelos de simulação, conceitos mínimos para a criação de pacotes R, descrição dos módulos que compõem a suíte DSSAT, descrição dos módulos que compõem o modelo genérico SimpleCrop e aplicações reais de acoplamentos de modelos encontrados na literatura. O entendimento de tais conceitos é necessário para a criação de um *middleware* que permita uma nova abordagem de desenvolvimento para modelos de cultura.

3. *MIDDLEWARE* PARA INTEGRAÇÃO DE MODELOS FORTRAN COM OUTRAS LINGUAGENS DE PROGRAMAÇÃO

3.1 RESUMO

Os modelos de simulação de culturas tem sido utilizados com grande sucesso em todo o mundo para aumentar a renda agrícola e reduzir os custos na produção. Mas está ficando cada vez mais difícil expandir e melhorar esses modelos devido à sua complexidade e ao nível de detalhe, tornando-se uma tarefa árdua e cara. A maioria dos modelos são/foram desenvolvidos em Fortran, linguagem que tem limitações na integração, na interoperabilidade e na visualização de dados. Por isto, este trabalho apresenta um *middleware* em C ++ que permite a integração de modelos de culturas desenvolvidos em Fortran com outras linguagens de programação. Com este *middleware*. os modelos desenvolvidos em Fortran podem acessar novas formas de aquisição de dados, como bases de dados e *WebServices*. Um estudo de caso é apresentado para ilustrar o *middleware* desenvolvido utilizando o modelo SimpleCrop por meio da criação de um pacote R, que pode facilmente manipular os dados de entrada e saída como qualquer objeto R. A execução do modelo, com isto, torna-se simples, como executar qualquer outro comando R.

3.2 INTRODUÇÃO

O Fortran tem sido usado no desenvolvimento de modelos matemáticos ao longo dos últimos 40 anos e, ainda, é muito difundido no desenvolvimento de software científico e de engenharia [43]. Um grande número de modelos de culturas desenvolvidos em Fortran são amplamente utilizados por pesquisadores, mas apresentam algumas limitações de acesso a tecnologias modernas. À medida que novos componentes são adicionados aos modelos de crescimento da cultura para expandir as suas capacidades, os modelos tornam-se cada vez mais complexos [9].

Muitas vezes, os modelos apresentam os resultados em arquivos com uma estrutura ASCII, assim os usuários podem perder horas na tentativa de interpretar as informações. O Fortran tem suporte limitado para visualização e interoperabilidade com outras linguagens e aplicações [43].

R é uma linguagem para estatística computacional e para visualização gráfica [44]. É uma implementação de código fonte aberto da linguagem S desenvolvida no *Bell Laboratories*. É uma ferramenta poderosa capaz de realizar modelos estatísticos complexos com uma grande variedade de interfaces gráficas de alta qualidade. O software base pode ser baixado diretamente do site R-project[3] e apresenta suporte para todos os sistemas operacionais modernos [44].

O sistema de pacote tem sido a chave do grande sucesso global do R, permitindo manter, de uma maneira fácil, coleções de funções, conjunto de dados e documentação e podendo ser distribuído

[3] http://cran.r-project.org

de forma simplificada. Os pacotes R podem ser carregados e descarregados em tempo de execução ocupando memória apenas quando são utilizados [44].

Na grande maioria dos vezes um programa é escrito em apenas uma linguagem de programação, mas em alguns casos e necessária a utilização de mais de uma linguagem de programação. Um exemplo é quando uma biblioteca é escrita em uma linguagem, mas é usada por outra linguagem. Para se ter acesso as bibliotecas e a mecanismos de baixo-nível suportados pela linguagem C pelo Fortran foi criado um padrão de declaração de variáveis e funções globais que são interoperáveis com a linguagem C [45].

A interoperabilidade entre R e C/C++ é limitada ao R realizar chamadas dos códigos escritos na linguagem C/C++, compilados como *Dynamic link library* (DLL) ou como *Shared Object* (SO). De fato, muitas das tarefas de computação intensiva exercidas pelo R são implementadas fazendo uso de DLL ou SO, tipicamente escritos em Fortran ou C/C++ [43]. O R fornece duas interfaces para códigos compilados em Fortran e C/C++ [46].

Segundo Coulouris et al. [47], o termo *middleware* se aplica a uma camada de software que fornece uma abstração de programação, assim como o mascaramento da heterogeneidade das redes, do hardware, dos sistemas operacionais e das linguagens de programação subjacentes.

O objetivo deste trabalho é criar um *middleware* genérico que permita a integração de modelos de código fonte legado com tecnologias modernas e outras linguagens de programação, removendo os arquivos de entrada e de saída, mantendo todos os dados em memória em tempo de execução, com poucas alterações no código fonte original, e para colocar em prova o *middleware* também é apresentado um pacote R que encapsule os modelos de simulação de código fonte legado.

3.3 *MIDDLEWARE*

Abordando a necessidade de realizar evoluções e integrações com outras linguagens de programação, modelos, base de dados, serviços web e novas formas de visualização dos dados e as limitações da linguagem do Fortran, foi desenvolvido um *middleware* em C++. A escolha da linguagem deve-se a interoperabilidade entre o Fortran e o C/C++ e a flexibilidade que a linguagem de programação C++ oferece em comunicar-se com outras linguagens de programação e bibliotecas.

O *middleware*, como pode ser visto na Figura 13, é responsável por intermediar a comunicação do Fortran com qualquer outra linguagem de programação. Ele realiza toda a gerência dos acessos a memória pelo modelo e é dividido em três partes principais: entrada e visualização dos dados, *middleware* C++ e o modelo desenvolvido em Fortran.

A tarefa de leitura dos dados é removida do Fortran e repassada para outra linguagem de programação que tenha interoperabilidade com C/C++ tais como, R, Java ou Phyton, possibilitando, assim, novas formas de aquisição dos dados além dos arquivos padrões do Fortran. Após a obtenção dos dados, eles são repassados por parâmetro para o *middleware*.

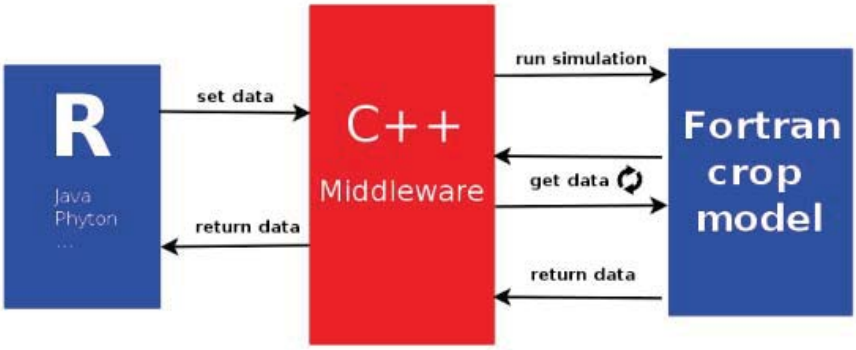

Figura 13. Estrutura do middleware.

O *middleware* C/C++ é responsável por manipular todos os dados, recebendo-os como parâmetro e os armazenando em memória e após inicializa o modelo. Ele disponibiliza um conjunto de funções que permitem que o modelo acesse os dados em memória quando necessário, ou os altere, ou inclua dados na memória.

O modelo é mantido praticamente na forma em que foi concebido sem ter o seu fluxo de execução alterado. Quando o modelo necessitar de algum dado ele irá realizar uma requisição para o *middleware* através de um módulo que contenha as interfaces de acesso, possibilitando, assim, a integração do modelo com o *middleware* através da chamada de uma função, com poucas alterações no código. Conforme o modelo vai gerando resultados de saída eles são incluídos na memória.

3.4 PACOTE RSIMPLECROP

Para avaliar a efetividade do *middleware* proposto e ter uma visão do seu potencial, desenvolveu-se um pacote com o modelo SimpleCrop [34], desenvolvido em Fortran 77 e compilado com o compilador Fortran Intel [48]. O pacote tem três principais objetivos: ler os dados e enviar para o *middleware*; executar o modelo; e, após o término da execução do modelo, apresentar os dados resultantes para o usuário.

Na execução do modelo são necessários cinco conjuntos de dados, planta, solo, clima, irrigação e controle da simulação. O pacote fornece funções especificas para a leitura dos dados de solo, planta e controle da simulação. Os dados de clima e irrigação são lidos em formato de `Data Frame`, convertidos em objetos R.

O trecho de código apresentado na Figura 14 é um exemplo de entrada dos dados de planta, que contém os mesmos parâmetros contidos no arquivo de entrada, o que torna mais fácil a sua manipulação, desta forma obtém-se um resultado em lista já no formato necessário pelo pacote para a execução do modelo. Também a leitura dos dados de clima e o controle da simulação junto com o comando para a execução do modelo.

Quando o comando para a execução do modelo é executado (`runSimpleCrop`), são passados como parâmetros o `Data Frame` de clima, a lista com os dados de solo, a lista com os dados

```
plant <- param.plant(plant.Lfmax = 12.0,plant.EMP2 =0.64,plant.EMP1 = 0.104,
                     plant.PD = 5.0,plant.nb = 5.3,plant.rm = 0.100,
                     plant.fc = 0.85, plant.tb = 10.0, plant.intot = 300.0,
                     plant.n = 2.0, plant.lai = 0.013, plant.w = 0.3,
                      plant.wr = 0.045,plant.wc = 0.255, plant.p1 = 0.03,
                     plant.f1 = 0.028)
weather <- read.table("weather.inp")
...
ctrl <- param.simCtrl(date.doyp = 121,date.frop = 3)
out<-runSimpleCrop(weather = weather, plant = plant, soil = soil,
                   irri = irrig, simCtrl = ctrl)
```

Figura 14. Comandos necessários para a execução do modelo.

de planta, um `Data Frame` com os dados de irrigação e a lista com os controles da simulação. O R realiza um tratamento transformando as variáveis de objetos R em vetores e envia para o *middleware*, o qual realiza o armazenamento dos dados em memória. Após, executa o modelo. Na Figura 15 é apresentado o diagrama do fluxo de execução do pacote.

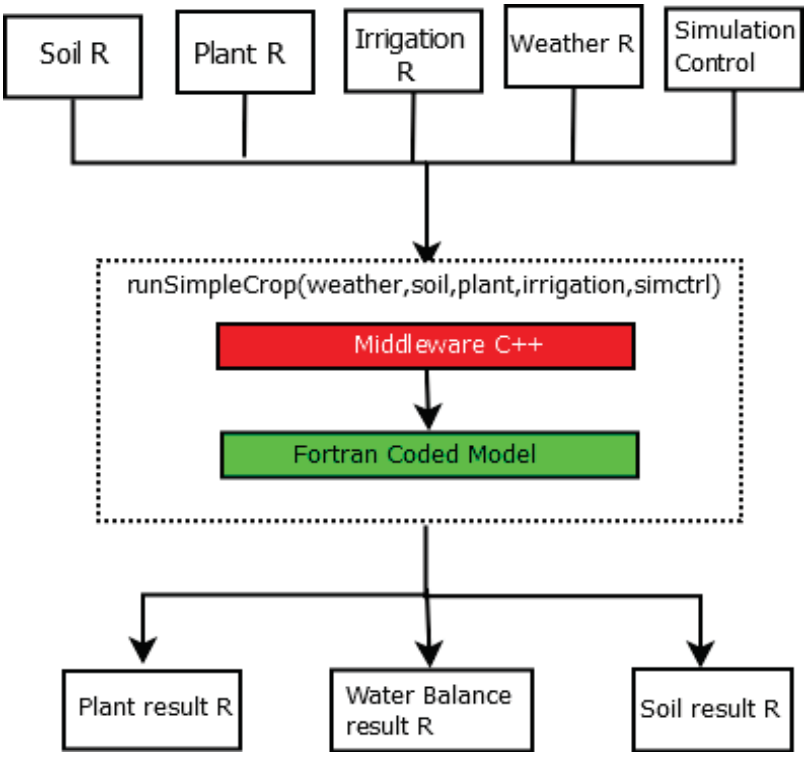

Figura 15. Estrutura do pacote RSimpleCrop.

Ao final da execução o pacote retorna ao usuário uma lista contendo três `Data Frames` com as saídas resultantes do modelo: crescimento da planta, balanço hídrico e informações solo (Figura 15).

Primeiramente é necessário a remoção do compilador Fortran da Intel, por ser um Software proprietário e necessita à aquisição de um licença por um compilador de código fonte aberto. Foi escolhido o compilador Gfortran[49] pertencente ao *GNU Compiler Collection*[50]. Para seu uso foi

preciso realizar poucas alterações no código fonte do modelo para ser compilado, como pode ser visto na Figura 16, que é dividida entre código original (A) e código alterado (B).

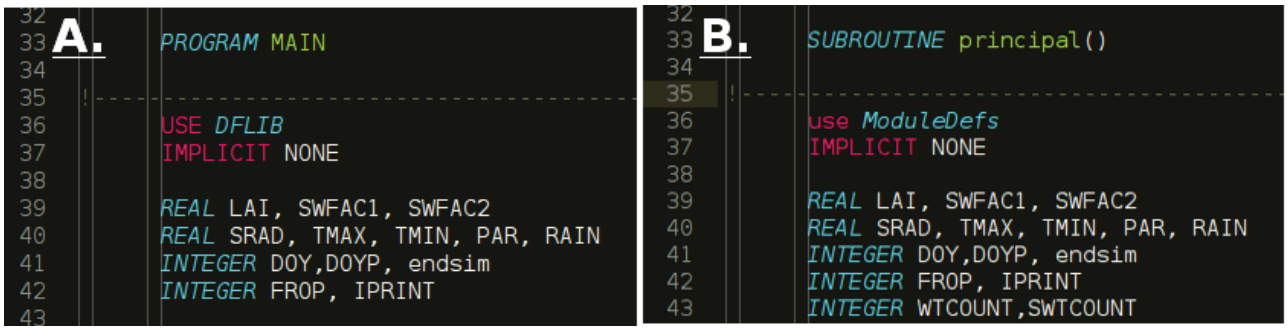

Figura 16. Diferenças entre as funções `Main.for` original (A) e código alterado (B).

Para realizar a compilação do modelo pelo Gfortran [49] é necessária a remoção da biblioteca `DFLIB`, como pode ser visto no código B na Figura 16 em comparação ao código A, por ser uma biblioteca do compilador Fortran Intel [48].

Com o intuito de a manter hierarquia no processo de compilação do modelo e do módulo de comunicação com o *middleware*, foram desenvolvidos dois arquivos de compilação (*Makefile*), um para criar um SO para sistemas operacionais baseados em Unix e outro para gerar DLL para sistemas operacionais baseados em Windows. No momento da instalação, o pacote R realiza a compilação do modelo e do *middleware* automaticamente, não sendo necessária nenhuma intervenção do usuário.

As comunicações entre o modelo e o *middleware* são feitas através de um módulo de interfaces (Figura 17) composto por funções que realizam a escrita e a leitura dos dados na memória do *middleware*.

Na Figura 17 é apresentado um exemplo de função usada para a leitura dos dados climáticos do *middleware* pelo modelo. A função é dividida em duas partes: a declaração da função, que segue o padrão com o nome da função e os seus parâmetros; a interface que contém uma função Fortran que será referenciada através do comando `bind` a uma função C/C++ pelo parâmetro `name`.

Para a integração do modelo com o *middleware* poucas alterações foram realizadas como pode-se ver na Figura 16: a função principal do modelo (A), foi alterado para uma rotina (B) que será chamada pelo middleware.

Na aquisição dos dados o modelo irá realizar a chamada de uma função do módulo de interfaces responsável por buscar os dados que são necessários na memória e retorná-los como parâmetros para o modelo, com poucas alterações no código fonte, conforme exemplo na Figura 18.

Conforme o exemplo da Figura 18, a leitura dos dados de planta pelo código original (A) é realizada da seguinte forma: o modelo busca e abre o arquivo de entrada e, após, realiza a leitura dos dados linha por linha. Já no código alterado (B) foi removida toda a manipulação de arquivos e, quando o modelo precisa de algum dado, ele realiza uma chamada de uma função do módulo de interfaces que irá acessar a memória através do middleware.

```
      SUBROUTINE READWTH (DATE,SRAD,TMAX,TMIN,RAIN,PAR,count)
            use, INTRINSIC :: iso_c_binding
!-----------------------------------------------------------------------

            interface
                    subroutine getwth_(date,srad,tmax,tmin,
     &        rain,par,wtcount) bind (c, name='getwth_')
                         use iso_c_binding
                 integer :: date
                 real :: srad,tmax,tmin,rain,par
                 integer :: wtcount
                    end subroutine getwth_
            end interface

      SAVE
      INTEGER DATE
      REAL SRAD,TMAX,TMIN,RAIN,PAR
      INTEGER count

      CALL getwth_(DATE,SRAD,TMAX,TMIN,RAIN,PAR,count)

      END SUBROUTINE READWTH
```

Figura 17. Exemplo de módulo de interfaces para integração com o middleware.

A.

```
      OPEN (2,FILE='PLANT.INP',STATUS='UNKNOWN')
      OPEN (1,FILE='plant.out',STATUS='REPLACE')

      READ(2,10) Lfmax, EMP2,EMP1,PD,nb,rm,fc,tb,intot,n,lai,w,wr,wc
     &     ,p1,sla
```

B.

```
          call READPLANT(Lfmax, EMP2,EMP1,PD,nb,rm,fc,tb,intot,n,lai,w,wr,
         &     wc,p1,sla)
```

Figura 18. Exemplo de código para leitura dos dados de planta, código original (A) e código alterado (B).

Originalmente o modelo realiza os cálculos de crescimento da planta, de balanço hídrico e de informações do solo, e vai escrevendo as saídas em seus respectivos arquivos. Como pode-se ver na Figura 19 (A), os dados de saída de crescimento da planta, após serem calculados, são escritos no arquivo através da função `WRITE` com um formato pré definido no código.

Já com a inclusão do *middleware* nas saídas, o modelo deixa de gravar nos seus arquivos de saída respectivos e passa a gravar os dados em memória. A Figura 19 (B) ilustra a alteração necessária para a inclusão do *middleware* no modelo para a escrita da saída do crescimento da planta, sendo necessário apenas a troca da função de escrita para a uma função do módulo de interfaces para a comunicação com o *middleware*.

```
COUNT = COUNT + 1                                    A.

WRITE(*,20) DOY,n,int,w,wc,wr,wf,lai
```

```
COUNT = COUNT + 1                                    B.

call WRITEPLANT(DOY,n,int,w,wc,wr,wf,lai,ctr)
```

Figura 19. Exemplo de código para escrita do resultado na memória, código original (A) e código alterado (B).

Após o middleware receber os dados de saídas do modelo os gravar em memória, ele os retorna como parâmetros e são transformados em objetos R. A Figura 20 apresenta o `Data Frame` de saída do crescimento da planta.

	Day of Year	Numer of Leafs Nodes	Accum Temp during Reprod (oC)	Plant Weight (g/m2)	Canopy Weight (g/m2)	Root Weight (g/m2)	Fruit weight (g/m2)	Leaf Area Index (m2/m2)
1	121	2.000	0.00	0.300	0.255	0.045	0.000	0.013
2	122	2.100	0.00	0.466	0.396	0.070	0.000	0.017
3	123	2.199	0.00	0.647	0.550	0.097	0.000	0.020
4	124	2.297	0.00	0.844	0.718	0.127	0.000	0.024
5	125	2.396	0.00	1.004	0.853	0.151	0.000	0.028
6	126	2.495	0.00	1.202	1.021	0.180	0.000	0.032
7	127	2.595	0.00	1.482	1.260	0.222	0.000	0.036
8	128	2.694	0.00	1.701	1.446	0.255	0.000	0.040
9	129	2.794	0.00	1.906	1.620	0.286	0.000	0.045

Figura 20. `Data frame` de saída do crescimento de planta.

3.5 RESULTADOS

A partir da execução do modelo pelo pacote tem-se todas as saídas do modelo como objetos R (Figura 20), permitindo uma fácil manipulação (inferência estatística, cálculos etc.) e a visualização dos dados de saída do modelo, por se tratar de objetos R em comparação com o formato padrão de arquivo texto de saída do modelo original.

A Figura 21 apresenta a representação gráfica dos resultados diários de crescimento da planta, os quais são: o número de nós da folha, o peso da planta,o peso da raiz e o índice de área foliar utilizando o pacote ggplot2[4] para a geração dos gráficos. Este exemplo não se destina a produzir

[4] http://ggplot2.org/

alguma análise estatística, mas apenas para demonstrar as novas possibilidades que o pacote permite para o modelo Fortran.

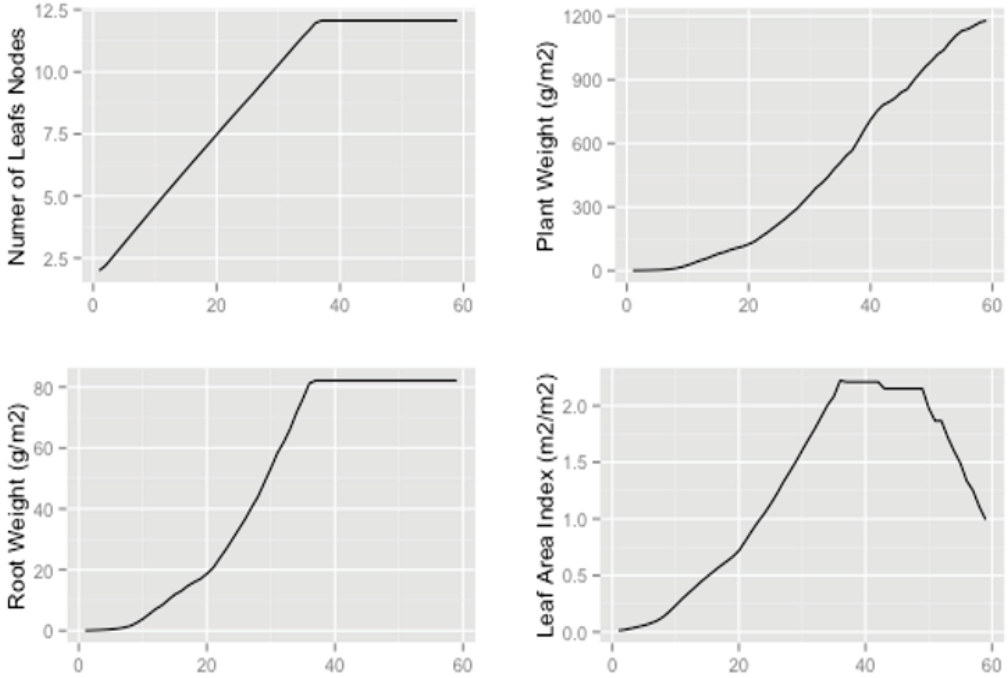

Figura 21. Exemplo de representação gráfica da saída do modelo.

3.6 CONSIDERAÇÕES

Motivado pelas limitações do Fortran e pela grande quantidade de modelos de simulação que foram e ainda estão sendo desenvolvidos, este trabalho apresentou o desenvolvimento de um middleware para a integração de modelos de código fonte legado com tecnologias atuais.

O *middleware* proposto permitiu plena integração entre Fortran e R, removendo completamente as manipulações dos arquivos pelo Fortran. Também permitiu esconder completamente toda a complexidade associada à execução do modelo.

O pacote R oferece uma maneira conveniente de manipulação de entrada e saída dos dados, permitindo o acesso a ferramentas como *WebServices* e base de dados. O pacote também permite realizar comparações entre cenários simulados pelo modelo com maior facilidade, além disso, permite a visualização gráfica.

Percebe-se que o pacote permite que o modelo seja executado em qualquer sistema operacional que tenha suporte ao R e o compilador de código fonte aberto Gfortran pertencente ao *GNU Compiler Collection*[45] sem que seja necessário conhecimentos aprofundados do sistema operacional por parte do usuário.

Por sua vez, o *middleware* permite não só a integração do modelo com o R mas sim com qualquer linguagem ou tecnologias que interaja com o C++, possibilitando novas evoluções sem ser necessário reescrever o modelo.

3.7 CONCLUSÃO

O *middleware* provou ser um ótimo recurso para integrar e reutilizar modelos de código fonte legado, trazendo grandes vantagens, tais como: o acesso a *WebServices*, o acesso a bases de dados e a visualização gráfica para o modelo, permitindo assim a sua modernização com alto grau de maturidade.

O pacote RSimpleCrop demonstrou que o *middleware* atingiu todos os objetivos proposto, permitindo um completo encapsulamento do modelo Fortran dentro de um pacote, escondendo toda a complexidade em torno da execução do modelo. Permitiu também a remoção completa da manipulação dos arquivos por parte do modelo Fortran e repassado para o R. Possibilita a integração do modelo Fortran com outras formas de aquisição de dados e a manipulação e a visualizações dos dados de saída.

4. *MIDDLEWARE* EM C++ PARA A EXTENSÃO DO DSSAT-CSM

4.1 RESUMO

O DSSAT-CSM (*Decision Support System for Agrotechnology Transfer Cropping System model*) é um sistema de apoio à tomada de decisão, reduzindo o tempo e os recursos humanos necessários para analisar e tomar decisões complexas. Foi desenvolvido para facilitar a aplicação de modelos de culturas numa abordagem sistemática nas pesquisas agronômicas. É composto por mais de 42 modelos, desenvolvidos quase que inteiramente em Fortran, linguagem que apresenta diversas limitações no que tange a aquisição de dados, a visualização, a integração com outras linguagens de programação, a dificuldade em acoplamento com outros modelos e a uma grande quantidade de chamadas de sistema para manipular arquivos. Com o objetivo de sanar estas limitações, desenvolveu-se um *middleware* responsável por gerenciar toda a entrada e a saída dos dados e os armazenando em memória por meio de uma estrutura genérica configurável em tempo de execução. O *middleware* disponibiliza três conjuntos de funções: `READ`,`GET` e `SET`. Esta solução proposta soluciona as limitações do Fortran sem alterar o fluxo de execução do DSSAT-CSM.

4.2 INTRODUÇÃO

As informações necessárias para auxiliar nas tomadas de decisões agrícolas estão aumentando rapidamente devido a grande demanda de produtos agrícolas e o aumento da pressão sobre a terra, a água e outros recursos naturais. A geração de novos dados através dos métodos tradicionais de pesquisa agronômica e sua publicação não são suficientes para atender as necessidades crescentes. Tradicionalmente, as experiências agronômicas são realizadas em pontos específicos no tempo e espaço, tornando os resultados locais e específicos da estação, caros e demorados. A menos que os novos dados e resultados provenientes da pesquisa sejam colocados em formatos que sejam relevantes e facilmente acessíveis, eles podem não ser usados de forma eficaz [9].

O DSSAT-CSM foi desenvolvido para facilitar a aplicação de modelos de culturas numa abordagem sistemática na pesquisa agronômica. Seu desenvolvimento inicial foi motivado pela necessidade de integrar o conhecimento sobre o solo, o clima, culturas e manejo na tomada das melhores decisões sobre a transferência da tecnologia de produção de um local para outro, onde o solo e o clima são diferentes [9].

O DSSAT-CSM auxilia na tomada de decisão, reduzindo o tempo e os recursos humanos necessários para analisar e tomar decisões complexas. Também fornece um quadro para a cooperação científica através de pesquisa para integrar novos conhecimentos [9].

Os dados mínimos necessários para a execução do DSSAT-CSM abrangem o local onde o modelo será operado (latitude, longitude, elevação, médias anuais de temperatura e sua amplitude), dados climáticos diários do período do ciclo de desenvolvimento da planta (radiação solar, temperatura

do ar máxima e mínima e precipitação), características do solo, condições inciais (cultura anterior, raiz, etc) e informações da cultura (tipo, nome do cultivar, data de plantio, profundidade, etc). Todos os modelos de culturas que compõem o DSSAT-CSM compartilham um formato comum de entrada e de saída e são similares em nível de detalhe, operando num passo de tempo diário [51].

Após averiguação, identificou-se que modelos presentes no DSSAT-CSM são quase que inteiramente desenvolvidos na linguagem Fortran, apresentando limitações nas formas de aquisição dos dados de entrada, na visualização dos dados de saída, na integração com outras linguagens de programação, na dificuldade em acoplamento com outros modelos, não utiliza JSON[5] (JavaScript Object Notation) e CSV (Comma Separated Values), gera excessiva utilização de arquivos temporários por não armazenar dados em memória, ocasionando uma grande quantidade de chamadas de sistema para operações de *Input/Output* (I/O).

O objetivo deste trabalho é criar um *middleware* que realize a gerência da entrada e saída (I/O) dos dados necessários para a execução dos modelos presentes no DSSAT-CSM. Além disto, visa permitir ao DSSAT-CSM novas formas de manipulação de formatos de dados de entrada e saída como JSON e CSV, integração com outras linguagens de programação, com poucas alterações no código fonte original, utilizando conceitos do *middleware* descrito na seção 3.3.

4.3 MIDDLEWARE

Após identificada as limitações na manipulações dos dados de entrada e saída e nas integrações com outras linguagens de programação pelo DSSAT-CSM, foi desenvolvido um *middleware* em C++ que realize a manipulação dos dados de entrada e saída permitindo novos formatos de arquivos como JSON e CSV, além dos formatos padrões. O *middleware* também permite a comunicação do DSSAT-CSM com outras linguagens de programação e bibliotecas.

Na Figura 22 é apresentada a estrutura do *middleware* para a extensão do DSSAT desenvolvido em C++, sendo responsável por gerenciar as manipulações dos arquivos de entrada e saída, sem alterar a forma de execução dos modelos. A estrutura pode ser divido em quatro parte principais: o DSSAT-CSM, o *middleware* C++, os arquivos de entrada e saída.

Como pode ser visto na figura 22, o *middleware* disponibiliza três conjuntos de funções: um conjunto de funções `READ` que são responsáveis por realizar as leituras dos arquivos de entrada para a memória, um conjunto de funções `GET` que são responsáveis por adquirir os dados armazenados em memória pela a função de leitura e um conjunto de funções `SET` que são utilizadas pelo modelo para armazenar dados em memória, também utilizada para gerar os arquivos de saída.

O conjunto de funções `READ` utiliza um arquivo de configuração para definir quais variáveis deverão ser lidas em tempo de execução, permitindo uma grande flexibilidade e a possibilidade de ser adicionada(s) ou removida(s) qualquer variável(eis) sem a necessidade de alteração do código fonte do *middleware*. Os arquivos são divididos por modelos devido as particularidades de cada modelo

[5] http://www.json.org/json-pt.html

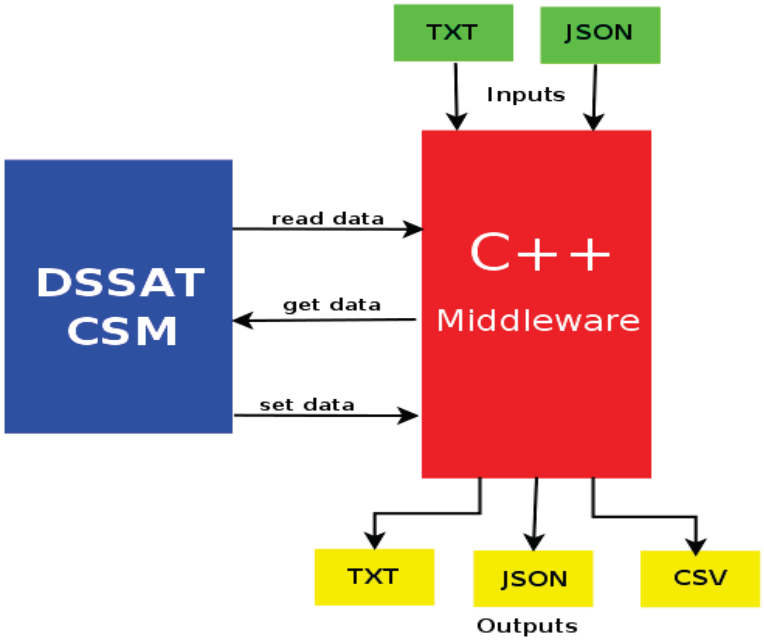

Figura 22. Estrutura do middleware para extensão do DSSAT desenvolvido em C++.

exceto os dados de experimento que, por serem o mesmo padrão para todos os modelos, há um só arquivo. Na Figura 23 é apresentado um trecho de um arquivo de configuração.

```
 96 *WEATHER
 97 #DAYS    365
 98 @HEADER
 99 !VARIABLE        TYPE    SIZE    LENGTH  READBY
100 INSI             string  1       5
101 LAT              real    1
102 LONG             real    1
103 ELEV             real    1
104 TAV              real    1
105 AMP              real    1
106 REFHT            real    1
107 WNDHT            real    1
108 CCO2             real    1
109
110 @DATA
111 !VARIABLE        TYPE    SIZE    LENGTH  READBY
112 DATE             integer 1096
113 SRAD             real    1096
114 TMAX             real    1096
115 TMIN             real    1096
116 RAIN             real    1096
117 DEWP             real    1096
118 WIND             real    1096
119 PAR              real    1096
```

Figura 23. Arquivo de configuração, seção de clima.

O arquivo de configuração (Figura 23) é dividido em: seção (*) o qual representa os arquivos de dados (*weather, cultivar, species, ecotype*), subseção(@) é a divisão das linhas do arquivo de dados e as constantes (#). O arquivo é composto por:

- `VARIABLE` nome da variável, utilizado como identificação não podendo haver nomes duplicados;
- `TYPE` é o seu tipo que pode ser `string`, `integer` ou `real`; *SIZE* define a quantidade de elementos que serão lidos;

- `LENGTH` utilizado para `strings`, define o comprimento;

- `READBY` define se a leitura será realizada por coluna (C) ou por linha (L), caso não seja declarado nenhuma das duas opções, a função por padrão irá realizar a leitura por linha;

Os dados lidos são armazenados na memória utilizando uma estrutura genérica chamada `unordered_map`[6]. É uma estrutura associativa que armazena elementos formados pela combinação de uma chave de identificação única e o valor que será armazenado, permitindo uma rápida recuperação de elementos individuais com base em suas chaves de identificação.

A função `GET` é responsável por buscar os dados lidos para a estrutura genérica na memória, recebe por parâmetro o nome da variável e/ou um inteiro (identificador único) que deseja-se buscar e a variável destino que pode ser carácter, inteiro, real, vetor de caracteres, vetor de inteiros, vetor de reais ou um índice de algum vetor, que irá receber o dado buscado. Na Figura 24 é apresentado um exemplo de uma função `GET`.

```
int getReal(char *VNAME, float * val) {
    std::string vname(VNAME), aux;

    ...

    std::unordered_map<std::string, std::string>::const_iterator got = dataGeneric.find (vname);
    if (got == dataGeneric.end())
        return 1;
    else
        *val = strtof (dataGeneric[vname].c_str(), NULL);
    fflush(stdin);
    return 1;
}
```

Figura 24. Exemplo de função GET para dados reais.

Na Figura 24 podemos observar um exemplo de função `GET`, responsável por buscar dados reais da estrutura genérica em memória. A função recebe o nome da variável (identificador) que está sendo buscada como um ponteiro para um vetor de caracteres que é convertido para uma `string` e um ponteiro do tipo `float` para retorno, realizando uma busca pelo identificador na estrutura genérica que, caso encontrado, atribui o dado ao ponteiro de retorno.

O conjunto de funções `SET` pode ser dividido em dois tipos: um para armazenar dados na estrutura genéria na memória e outro para realizar as escritas dos dados nos arquivos de saídas utilizando um arquivo de configuração. na Figura 25 é demonstrado um exemplo da função `SET` para armazenar dados na estrutura genérica em memória e na Figura 26 é apresentado o arquivo de configuração de saídas para o uso da função `SET` para gerar os arquivos de saída.

Na figura 25 podemos observar um exemplo de função `SET`, responsável por armazenar dados reais na estrutura genérica em memória. Afunção também recebe o nome da variável (identificador) que será armazenada na memória como um ponteiro para um vetor de *characters* que é convertido para uma *string* e um ponteiro do tipo *float* para o valor que será armazenado, caso o identificador já exista na memória o valor existente será atualizado.

[6] http://www.cplusplus.com/reference/unordered$_map/unordered_map/$

```
int setRealMem(char *VNAME, float *val) {
    std::string vname(VNAME);
    ...
    std::transform(vname.begin(), vname.end(), vname.begin(), ::toupper);
    dataGeneric[vname] = std::to_string(*val);
    return 1;
}
```

Figura 25. Exemplo de função SET para dados reais.

Para utilização da função SET na geração dos arquivos de saída desenvolveu-se um arquivo de configuração (Figura 26) que permite a formatação da saída e quais variáveis serão escritas em tempo de execução.

```
58 *EVALUATE
59 $SEP              ,
60 @HEADER           Y
61 !VARIABLE         OUTPUT          FORMAT          TFORMAT
62 NAME              Y               %1s\t\t\t
63 MODEL             Y               %85s\n\n
64 @DATA             Y
65 $TITLE            Y
66 !VARIABLE         OUTPUT          FORMAT          TFORMAT
67 RUN               Y               %4i             %4s
68 EXCODE            Y               %11s            %11s
69 TN                Y               %6i             %6s
70 RN                Y               %3i             %3s
71 CR                Y               %3s             %3s
72 ADAPS             Y               %8s             %8s
73 ADAPM             Y               %8s             %8s
74 PD1PS             Y               %8s             %8s
75 PD1PM             Y               %8s             %8s
76 PDFPS             Y               %8s             %8s
77 ...
```

Figura 26. Exemplo de arquivo de configuração para uso da função SET na geração de arquivos de saída.

O arquivo de configuração de saída, apresentado na Figura 26 é dividido em seção (*) que representa qual é o arquivo de saída, subseção (@) informa as seções que compõem o(s) arquivo(os) de saída, permitindo que seções inteiras sejam removidas em tempo de execução e as variáveis globais($) que podem afetar todo o arquivo de saída ou só uma subseção. O arquivo é composto por:

- `VARIABLE`: nome da variável que será escrito no arquivo de saída como cabeçalho e também servindo como identificador único;
- `OUTPUT`: define se a variável será escrita (Y) ou não (N);
- `FORMAT`: define a formatação dos dados que serão escritos no arquivo de saída, utilizando o padrão da linguagem C;
- `TFORMAT`: define a formatação do nome da variável que será escrito no arquivo de saída como cabeçalho, utilizando o padrão da linguagem C;

Além de permitir novos formatos para os arquivos de dados como, por exemplo, o JSON (Figura 22), o *middleware* possibilita a integração com sistemas de informação, *WebServices*, dentre outros. Também possibilita a integração com outras linguagens de programações, bibliotecas e que tenham interoperabilidade com a linguagem C/C++, permitindo facilmente o acoplamento com outros modelos de simulações.

Para o uso do *middleware* poucas alterações nos módulos do DSSAT-CSM são necessárias, bastando a chamada de uma função, simplificando o código e não alterando o fluxo e as formas de execução dos modelos.

4.4 COMPILAÇÃO

Originalmente o DSSAT-CSM já vem preparado para a compilação e a execução no sistema Operacional Windows[7] com o compilador da Intel Ifort [48], sendo, portanto, necessário realizar a adaptação para a compilação e a execução em sistemas operacionais baseados em Unix.

Para realizar esta tarefa, primeiramente, é necessário baixar o código fonte do DSSAT-CSM presente no repositório do DSSAT no Github[8]. Após descompactar o código fonte, é preciso separá-lo em duas pastas: uma para o código fonte (`Source`) e outra para os dados (`DSSAT46`), como pode ser visto na Figura 27.

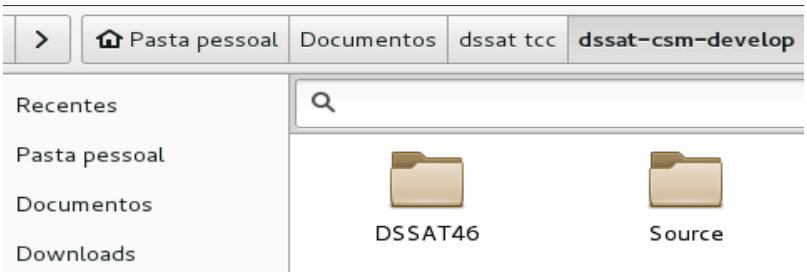

Figura 27. Estrutura de pastas utilizada pelo DSSAT-CSM.

Após separado as pastas (Figura 27), é necessário alterar o arquivo de perfis **DSSATPRO.v46** para a extensão **I46**. Este arquivo indica os caminhos onde serão encontrados os executáveis dos modelos e os diretórios dos arquivos de entrada. Também é necessária a alteração do arquivo nas linhas que contenham **"C:"** para **".."** e nos lugares onde contém **"\"** para **"/"**, como pode ser visto na Figura 28 onde o trecho do arquivo de perfis *DSSATPRO.v46* alterado para `DSSATPRO.l46`.

Em seguida é necessário alterar o código para ser executado em sistemas operacionais baseados em Unix. Há dois pontos no código fonte que tratam as diferenças entre os sistemas operacionais. Na Figura 29 são ilustradas as alterações realizadas no arquivo `ModuleDefs.for`, onde contém a variável que define para qual sistema operacional será gerado o executável, responsável, por exemplo, por qual arquivo de perfis que será usado. Para alterar a variável de ambiente basta alterar a linha comentada.

[7] https://www.microsoft.com/pt-br/windows/
[8] https://github.com/orgs/DSSAT/

```
DCG  C: \DSSAT46\TOOLS\GENCALC GENCALC2.EXE
DGL  C: \DSSAT46\TOOLS\GLUE GLUE.BAT
DPM  C: \DSSAT46                                    v46
DDE  C: \DSSAT46
TOG  C: \DSSAT46\TOOLS\GBUILD GBUILD.EXE
```

```
DCG  .. /DSSAT46/TOOLS/GENCALC GENCALC2.EXE
DGL  .. /DSSAT46/TOOLS/GLUE GLUE.BAT
DPM  .. /DSSAT46/                                   l46
DDE  .. /DSSAT46/
TOG  .. /DSSAT46/TOOLS/GBUILD GBUILD.EXE
```

Figura 28. Trecho do arquivo de perfis DSSATPRO.v46 alterado para a execução em sistemas operacionais baseados em Unix, DSSATPRO.l46.

```
35      MODULE ModuleDefs
36  !   Contains defintion of derived data types and constants which are
37  !   used throughout the model.
38      SAVE
39  !-----------------------------------------------------------------------
40  !   Change this line to switch between Windows and Linux compilers
41  !   Operating system
42  !    CHARACTER(LEN=5), PARAMETER :: OPSYS = 'WINDO'   !DOS, Windows
43  |    CHARACTER(LEN=5), PARAMETER :: OPSYS = 'LINUX'   !Linux, UNIX
44  !-----------------------------------------------------------------------
```

Figura 29. Definição da variável de ambiente que define o sistema operacional no arquivo ModuleDefs.for.

O próximo arquivo que deve ser alterado é o `CRSIMDEF.for`, que define se ele irá utilizar barra ou contra-barra dependendo do sistema operacional. Na Figura 30 é ilustrada a alteração que foi realizada para o uso da barra, bastando trocar a linha comentada com o respectivo sistema operacional.

```
5
6       MODULE CRSIMDEF
7
8       SAVE
9
10  !    CHARACTER(LEN=1),PARAMETER::SLASH = '\' !DOS, Windows
11  |    CHARACTER(LEN=1),PARAMETER::SLASH = '/' !Linux, Unix
12
13      END MODULE CRSIMDEF
```

Figura 30. Definição do uso de barra ou contra-barra dependendo do sistema operacional no arquivo CRSIMDEF.for.

Para a compilação com o compilador Gfortran [49] é necessário alterar a sintaxe de funções em alguns trechos do código, pois há diferenças em algumas funções entre o compilador Ifort da Intel [48] e o Gfortran. Na Figura 31 são ilustradas as alterações necessárias para o uso da função `IACHAR`, pois no Gfortran, no aquivo `CSUTS.for`, a função recebe um carácter como parâmetro, já no compilador Ifort a função recebe um vetor de caracteres, a variável `newchar` é alterada de um vetor com tamanho 10 para um com tamanho 1.

No arquivo `SPSUBS.for` para a compilação com Gfortran é necessário alterar a função `WRITE`, como pode ser observado na Figura 32. Nela foram alterados os parenteses devido que o Gfortran ter a sua sintaxe um pouco diferente quando comparado ao compilador da Ifort da Intel.

```
1981        CHARACTER (LEN=*)  charnum
1982        !CHARACTER (LEN=10) newchar
1983        CHARACTER (LEN=1) newchar
```

Figura 31. Alterações realizadas no arquivo CSUTS.for para a compilação com o compilador Gfortran [49].

```
127         ! WRITE (LUN,122)("ES",L,"D",L=1,9, "        ES10D    RWUD")
128           WRITE (LUN,122) ("ES",L,"D",L=1,9), "       ES10D    RWUD"
129   122     FORMAT(9("    ",A2,I1,A1),A25)
```

Figura 32. Alterações realizadas no arquivo SPSUBS.for para a compilação com o compilador Gfortran [49].

Após realizadas as alterações no código fonte, o DSSAT-CSM estará apto para ser compilado e executado em sistemas operacionais baseados em Unix. Para facilitar a compilação é recomendado a criação de um arquivo de compilação (`makefile`) que irá automatizar o processo de compilação e de criação do executável.

4.5 ADAPTAÇÃO DO DSSAT-CSM PARA USO DO *MIDDLEWARE*

Poucas adaptações são necessárias para integrar o DSSAT-CSM com o *middleware* C++, permitindo que sejam realizadas novas evoluções sem ter que alterar o fluxo de execução original dos modelos. Para a utilização do *middleware* foram alteradas as funções que realizam as leituras dos dados de cultivares, de ecótipo, de espécies, de clima e de experimento.

O DSSAT-CSM não acessa diretamente as funções `GET` e `SET` disponíveis no *middleware*, visando esconder toda a complexidade do C/C++ dos programadores Fortran. Por isto desenvolveu-se um módulo de interface em Fortran que realiza as chamadas das funções do *middleware*. Na Figura 33 é apresentada a estrutura de mapeamento das funções `GETS` e `SETS`.

```
type csm_io_type
     contains
       procedure :: set_io_val_real
       procedure :: set_io_val_int
       ...

       procedure :: get_io_val_real
       procedure :: get_io_val_int
       ...

       generic   :: set => set_io_val_real,set_io_val_int,&
            ...
       generic   :: get => get_io_val_real,get_io_val_int,&
            ...
end type csm_io_type
```

Figura 33. Estrutura de mapeamento das funções GETs e SETs.

Na estrutura `csm_io_type` (Figura 33) são declarados os nomes das funções de interface com o *middleware* que estarão disponíveis para o programador Fortran, realizando o mapeamento das funções que serão chamadas apenas como `SET(parâmetros)` ou como `GET(parâmetros)`. A definição de qual das funções será chamada é realizada através dos parâmetros que a função recebe.

O módulo de interfaces também contém as funções de interface com o *middleware*, com o seu nome declarado como uma `procedure` na estrutura `csm_io_type` (Figura 33). Para exemplificar uma função de interface com o C++, na Figura 34 é apresentada uma função `GET` para um valor do tipo real.

```
subroutine get_io_val_real(ioset,vname,val)

  use, INTRINSIC ::iso_c_binding
  character(len=*),intent(in)          :: vname
  real,intent(out)                     :: val
  class(csm_io_type)                   :: ioset
  character*100 aux
interface
        subroutine get_real(vname,val)&
        bind(C, name='getReal')
        use, intrinsic :: iso_c_binding
        character(kind=c_char), dimension(*) :: vname
        REAL :: val
        end subroutine get_real
end interface
call get_real(aux,val)

end subroutine get_io_val_real
```

Figura 34. Exemplo de função interface GET para acesso ao middleware.

A função `get_io_val_real`, como pode ser visto na Figura 34, recebe como parâmetro uma variável do tipo `csm_io_type` que é utilizada no mapeamento das funções baseadas nos seus parâmetros, um vetor de caracteres que define o nome da variável (identificador) que será buscada e uma variável do tipo real que irá receber o valor buscado em memória.

Para usar o módulos de interfaces basta carregá-lo com `use csm_io`, que contém uma variável do tipo `csm_io_type` que é usada para realizar todas as chamadas das funções de interface com o *middleware*. Para usar as funções basta chamar **`call csminp%get(parâmetros)`** ou **`call csminp%set(parâmetros)`** e a escolha de qual função chamar é definida através dos parâmetros passados.

O DSSAT-CSM não mantém os dados em memória, utilizando um sistema de arquivos temporários para armazenar os dados e realizar as trocas de informações entre os módulos. Todo e qualquer dado lido é escrito em seguida em um arquivo para ser utilizado pelos modelos posteriormente, resultando em uma excessiva quantidades de chamadas de sistema para manipulação de arquivos.

4.5.1 Cultivar

O módulo de Cultivar é responsável por realizar a leitura da linha, que corresponde com o identificador de cultivar definido no arquivo de experimentos, e escrevê-la em um arquivo temporário pronto para os modelos utilizarem os dados lidos. Na Figura 35 são apresentadas as alterações necessárias para utilizar a função de leitura disponibilizada pelo *middleware*.

```
CALL READCULTIVAR(FILEGG, MODEL,VARNO,ISECT)
IF(ISECT .EQ. 10) RETURN

IF(ISECT .EQ. 9) CALL ERROR (ERRKEY,1,FILEG,0)
IF (ISECT .EQ. 0) CALL ERROR (ERRKEY,2,FILEG,LINVAR)
```

Figura 35. Alterações realizadas no módulo de cultivar para uso do middleware.

As alterações realizadas no módulo de cultivar (Figura 35) foram a adição da chamada para a função de leitura dos dados de cultivar do *middleware*, que busca as variáveis que serão lidas no arquivo de configuração (Figura 23) na seção de cultivar, e o armazenando na estrutura genérica em memória. Caso a leitura para o modelo escolhido não esteja disponível no *middleware*, o módulo continua para a leitura padrão.

Os dados lidos ficarão disponíveis na memória para serem acessados e/ou alterados a qualquer momento que seja necessário. Para acessar os dados de cultivar lidos pelo *middleware*, foi alterado no código fonte todos os lugares que buscavam os dados nos arquivos temporários e adicionado a função `GET`. Na Figura 37 é demonstrado o uso da função `GET` para buscar os dados de cultivar.

```
call csminp%get("ECO#",ECONO)
call csminp%get("SLAVR",SLAVAR)
call csminp%get("SIZLF",SIZELF)
call csminp%get("XFRT",XFRUIT)
call csminp%get("THRSH",THRESH)
call csminp%get("SDPRO",SDPRO)
call csminp%get("SDLIP",SDLIP)
```

Figura 36. Função GET para buscar os dados de cultivar no arquivo DEMAND.for.

Ao invés de realizar a leitura dos dados de cultivar oriundos dos arquivos temporários o modelo passa a buscar os dados em memória com a função `GET`, como pode ser visto na Figura 37,

passando como parâmetro o nome (identificador) da variável que se deseja buscar e que deve estar presente no arquivo de configuração e a variável que irá receber o retorno da função GET.

4.5.2 Ecótipo

A leitura dos dados de ecótipo são realizados pelos modelos, onde cada modelo tem sua própria função de leitura que busca a linha que contém o código de ecótipo definido no arquivo de cultivares. Em cada lugar do modelo que necessite dos dados de ecótipo são lidos do arquivo, dificultando, assim, a manutenção e evoluções nas leituras. Na Figura 37 são demonstradas as alterações necessárias para utilizar o *middleware*, que visam centralizar todas as leituras e permitir o uso do armazenamento em memória através da estrutura genérica.

```
CALL READECO(FILEGC,FOUND, CONTROL%MODEL,ECONO)
ECOTYP = '        '
LNUM = 0
call csminp%get("ECO#",ECOTYP)
call csminp%get("KCAN_ECO",KCAN_ECO)
```

Figura 37. Alterações para a manipulação do arquivo de ecótipo pelo middleware.

Como apresentado na Figura 37, com o uso do *middleware* a leitura do arquivo de ecótipo é realizada apenas uma vez, e é armazenada na estrutura genérica em memória com as variáveis mapeadas através do arquivo de configurações (Figura 23). É necessário encontrar a primeira função do DSSAT-CSM que necessita dos arquivos de ecótipo pois ela será responsável por chamar a função de leitura do *middleware*. Em outros lugares que utilizem os dados de ecótipo basta pegar através de um GET e não mais por meio da leitura do arquivo.

Para buscar os dados de ecótipo na estrutura genérica através da função GET é necessário passar por parâmetro o nome da variável (identificador) a qual será buscada e a variável de retorno, como demonstrado na Figura 37, removendo, assim, toda e qualquer leitura do arquivo de ecótipo do Fortran e delegando está tarefa ao *middleware* C++.

4.5.3 Especies

A leitura dos dados de especies é muito semelhante à leitura dos dados de ecótipo (seção 4.5.2) onde cada modelo tem a sua função de leitura dos dados. Diferentemente do arquivo de ecótipo, o arquivo de espécies é lido por inteiro, sendo definido por seções, podendo ser lido por linhas e/ou colunas.

Em cada função que necessite dados de espécies, é realizada uma nova leitura no arquivo de espécies em busca da seção que contenha os dados necessários para a função. NA Figura 38 são apresentadas as alterações realizadas para utilizar o *middleware* para leitura dos dados de espécies e para o armazenamento em memória.

```
      CALL READSPE(FILECC,FOUND, "CRGRO")
C-----------------------------------------------
C READ PHOTOSYNTHESIS PARAMETERS ***************
C-----------------------------------------------
      SECTION = '!*PHOT'
      call csminp%get("KCAN",KCAN)
      call csminp%get("KC_SLOPE",KC_SLOPE)
      IF (KC_SLOPE .LT. 1.E-6) KC_SLOPE = 0.1
```

Figura 38. Alterações para a manipulação do arquivo de especies pelo middleware, e uso da função GET para acessar a estrutura genérica.

A função de leitura disponível no *middleware* (Figura 38) realiza a aquisição dos dados de espécies e os armazena na estrutura genérica em memória utilizando o arquivo de configuração para mapear as variáveis. A leitura é realizada apenas uma vez, reduzindo assim a quantidade de chamadas de sistema para manipulação de arquivo. A leitura deve ser chamada pela primeira função do DSSAT-CSM que necessite dos dados de espécies.

Após os dados serem armazenados em memória na estrutura genérica, eles estarão aptos a serem acessados da mesma forma que os dados de outros arquivos, através da função `GET`, passando como parâmetro o nome da variável (identificador) e a variável de retorno, como pode ser visto na Figura 38.

4.5.4 Clima

Originalmente o módulo de clima realiza a leitura de todos os dados climáticos presentes no arquivo e os armazenando em vetores. Logo após, faz uma busca nos vetores para encontrar o índice que contenha a data anterior a data de inicio da simulação, definida no arquivo de experimentos. Caso a data não seja encontrada ou os dados acabem antes que a simulação finalize, o DSSAT-CSM realiza buscas de outro arquivo que contenha os dados climáticos.

Com o uso do *middleware*, a leitura dos dados de clima são realizados por blocos. O primeiro bloco de dados a ser lido tem o seu tamanho definido no arquivo de configuração e, caso não seja definido, o *middleware* lê por padrão 365 dias, começando na data anterior a data de inicio da simulação. Os blocos subsequente são lidos com um tamanho de 50 a partir da próxima data (`YRDOYWY`)

da última lida pelo primeiro bloco. Na Figura 39 pode ser vista esta chamada da função de leitura dos dados climáticos.

```
call csminp%get(YRDOY,"DATE",DATE)

IF(DATE .EQ. -99)THEN
  CALL READWEATHER(FILEWW,YRDOYWY,CONTROL%YRDOY,CONTROL%YRSIM,
    CONTROL%MULTI,END_OF_FILE,FILEX,CONTROL%MODEL,1)
    IF(END_OF_FILE .EQ. 1)THEN
        YRSIMMY = INCYD(YRDOY,-1)
        CALL YR_DOY(YRDOY, YEAR, DOY)
        CurrentWeatherYear = YEAR
        ...
```

Figura 39. Trecho de código responsável por realizar a leitura dos dados climáticos em blocos.

Antes de realizar a próxima leitra de bloco, a função (Figura 39) verifica se encontra a data (YRDOY) na estrutura genérica em memória, a qual representa a linha do arquivo que contém os dados climáticos necessários pelo modelo. Caso não encontre a função de leitura, é chamada com o a nova data (YRDOY) que será buscada realizando assim a leitura de mais um bloco de tamanho 50 para a estrutura genéria. Caso o arquivo de clima acabe antes de atingir o tamanho do bloco, outro arquivo de dados climáticos é encontrando pelo DSSAT-CSM e passado para a função que prossegue a leitura até atingir o tamanho do bloco. Se a data for encontrada, prossegue para as funções `GET`, como pode ser visto na Figura 40.

```
call csminp%get(YRDOY,"DATE",YRDOYW)
CALL YR_DOY(YRDOYW, YEARW, DOYW)
call csminp%get(YRDOY,"SRAD",SRAD)
call csminp%get(YRDOY,"TMAX",TMAX)
call csminp%get(YRDOY,"TMIN",TMIN)
call csminp%get(YRDOY,"RAIN",RAIN)
call csminp%get(YRDOY,"DEWP",TDEW)
call csminp%get(YRDOY,"WIND",WINDSP)
call csminp%get(YRDOY,"PAR",PAR)
call csminp%get(YRDOY,"RHUM",RHUM)
call csminp%get(YRDOY,"VAPR",VAPR)
call csminp%get(YRDOY,"DCO2",DCO2)
```

Figura 40. Alterações para a manipulação do arquivo de especies pelo middleware, e uso da função GET para acessar a estrutura genérica.

A função `GET` é utilizada para buscar os dados climáticos (Figura 40) e recebe como parâmetros a data (`YRDOY`) que corresponde o dia do ano em conjunto com o nome da variável formando,

assim, a identificação de qual dado é buscado. Originalmente, o DSSAT-CSM utiliza um formato de ano representado com 2 dígitos e dia do ano. Com o uso do *middleware* é possível realizar a leitura tanto de anos com 2 ou 4 dígitos. O outro parâmetro passado para a função é a variável de retorno.

Os arquivos de clima muitas vezes são compostos com dados observados de muitos anos, resultando em arquivos extensos e com muitos dados. A estrutura de leitura em blocos permite que seja lido apenas os dados necessários, em comparação com a função de leitura original do DSSAT-CSM a qual realizava a leitura de todos os dados contidos nos arquivos de clima.

4.5.5 Experimentos

O arquivo de experimentos é responsável pela configurações da execução, de qual modelo utilizar, de quais os tratamentos que serão simulados, de qual cultivar e de qual ecótipo ele irá buscar e pelo arquivo que contém os dados climáticos e as informações sobre fertilizantes, irrigação e controles da simulação.

Cada seção do arquivo é divido em uma função específica de leitura, que lê, trata e armazena nos arquivos temporários. Posteriormente, serão lidos por outros módulos que necessitem dos dados de configuração que estão presentes no arquivo de experimento, resultando numa excessiva quantidade de chamadas de sistema para manipulação de arquivos.

A leitura do *middleware* permite centralizar todas as leituras em apenas uma função, como pode ser visto na Figura 41, configurável em tempo de execução através de um arquivo de configuração semelhante aos arquivos de configuração dos modelos.

```
120 C-----------------------------------------
121       FILELS = 'EXP.LST'
122
123       call READXFILE(FILEX)
124       ALN = ""
125 C-----------------------------------------
```

Figura 41. Função para leitura do arquivo de experimento.

A função de leitura (figura 41) utiliza o arquivo de configuração para mapear quais variáveis serão lidas, o tipo de leitura (linha ou coluna) e armazenando os dados na estrutura genérica em memória, eliminando, assim, os arquivos temporários utilizados.

Após lidos, os dados do arquivo de experimento estarão disponíveis para serem acessados e/ou alterados por qualquer módulo que os necessite. Como nos outros módulos é utilizada a função `GET` para acessar os dados armazenados em memória na estrutura genérica.

4.6 RESULTADOS

Com o uso do *middleware* para manipular os dados de entrada e saída, foi permitido o armazenamento em memória, a redução da quantidade de chamadas de sistema para manipular dados e a remoção dos arquivos temporários. Além disto, permitiu ao DSSAT-CSM trabalhar com novos formatos de entrada e saída, como pode ser visto na Figura 42, onde são apresentados exemplos de saídas do PlantGro em formatos Json e CSV, permitindo a formatação em tempo de execução dos arquivos de saída padrões.

```
*GROWTH ASPECTS OUTPUT FILE

*DSSAT Cropping System Model Ver. 4.6.1.004 -devel

*RUN   1        : BRAGG  1/10              CRGRO0
MODEL           : CRGRO046 - Soybean
EXPERIMENT      : EBPF5701 SB EBPF0001WH PASSO FUND
DATA PATH       :
TREATMENT  1    : BRAGG  1/10              CRGRO04

Root Dens. (cm/cm3) by soil depth (cm):
0-5    5-15   15-20   20-30   30-40   40-50   50-6
@YEAR,DOY,DAS,DAP,L#SD,GSTD,LAID,LWAD,SWAD,GWAD,Rw
1957,274,1,0,0.0,0,0.000,0,0,0,0,0,0,0,0.0,0.000,0
```

```
{
  "GROWTHOUT": [
    {
      "DAY": [
        {
          "CHTD": 0.0,
          "CWAD": 0,
          "CWID": 0.0,
          "DAP": 0,
          "DAS": 1,
          "DOY": 274,
          "EWSD": 0.0,
          "G#AD": 0,
          "GSTD": 0,
```

Figura 42. Novas saídas possibilitadas pelo middleware, Json E CSV.

4.7 CONSIDERAÇÕES

Através do uso do *middleware* obteve-se uma redução na quantidade de chamadas de sistema para a manipulação dos arquivos de dados, removendo os arquivos temporários que originalmente eram utilizados para armazenamento e para a troca de dados entre os módulos, possibilitando o armazenamento dos dados em memória.

Possibilitou a centralização das funções de leituras configuradas em tempo de execução pelos arquivos de configuração, permitindo a adição ou remoção de variáveis e dos formatos de leituras sem ser necessário a alteração do código fonte.

Proporcionou, ainda, mais dois formatos de arquivos de saída (JSON e CSV), formatados em tempo de execução através de um arquivo de configuração, permitindo ao usuário escolher apenas as informações que ele precisa.

Mesmo com todas alterações realizadas no DSSAT-CSM, o seu fluxo de execução e a forma de execução por linha de comando se mantiveram inalterados, sem ter grandes impactos tanto para os usuários como para os programadores Fortran.

Atualmente há três modelos do DSSAT-CSM que utilizam parcialmente o *middleware*: o modelo WHAPS que utiliza as funções de clima, cultivar, ecótipo e espécies; o modelo MZCER que utiliza

a função de clima; e o modelo SALUS que faz uso das funções de clima e cultivar. Já o modelo CROP-GRO teve todas as leituras dos seus arquivos removidos do Fortran e repassados para o *middleware* (clima, cultivar, ecótipo, especies, clima e experimento).

4.8 CONCLUSÃO

O middleware provou ser uma ótima ferramenta para realizar as manipulações de entrada e saída dos dados do DSSAT-CSM, permitindo novos formatos de dados, como o Json e o CSV, possibilitando, assim, uma fácil integração com sistemas de informações e facilitando o acoplamento com outros modelos. Também permite adicionar novos formatos de arquivos de entrada e saída sem ser necessário a alterações nos código fonte do DSSAT-CSM.

5. CONSIDERAÇÕES FINAIS

A utilização de modelos de simulação tem sido utilizados com grande sucesso em todo o mundo para aumentar a renda agrícola e reduzir os custos na produção e os recursos humanos necessários para analisar e tomar decisões complexas.

Muitos modelos que foram desenvolvidos em Fortran no passado ainda são muito utilizados nos dias atuais e novos modelos ainda continuam sendo desenvolvidos em Fortran. O Fortran é uma linguagem muito limitada fazendo muito uso de arquivos textos, tanto para entrada como para saída de dados. Além disso, possui uma integração muito limitada ou inexistente com outras linguagens ou tecnologias.

Motivado por estas limitações do Fortran e a grande quantidade de modelos de simulação que foram e ainda estão sendo desenvolvidos, este trabalho apresentou o desenvolvimento de um *middleware* em C++ para a integração de modelos de código fonte legado com tecnologias atuais. O *middleware* C++ proposto permitiu plena integração entre o Fortran e o R, permitindo o completo encapsulamento do modelo dentro de um pacote R, removendo completamente as manipulações dos arquivos pelo Fortran, permitindo, também, ´´esconder" completamente toda a complexidade associada à execução do modelos.

Visto o sucesso que o *middleware* C++ obteve na integração de um modelo em Fortran com outras tecnologias, na segunda etapa deste trabalho desenvolveu-se um *middleware* C++ para a extensão do DSSAT. Este *middleware* foi responsável pela tarefa de aquisição dos dados e de escrita dos arquivos de saída.

O *middleware* possibilitou que as funções de leitura dos dados fossem mais dinâmicas, podendo adicionar ou remover variáveis ou tipos de formatos de leituras em tempo de execução, sem ser necessária a alteração da função de leitura, possibilitando, ainda, que os arquivos de saída também pudessem ser formatados em tempo de execução. Também proporcionou uma redução considerável na quantidade de chamada de sistema para a manipulação dos arquivos.

Para que o programador Fortran não tenha acesso ou necessidade de conhecimento da complexidade das funções em C++, desenvolveu-se um conjuntos de funções de interfaces genéricas para acessar o *middleware*.

Por fim, o *middleware* permitiu com que novos formatos de entrada e saída de dados pudessem ser adicionados ao DSSAT-CSM de uma forma fácil e menos invasiva possível. Também possibilitou que o DSSAT-CSM fosse integrado com sistemas de informação e com outras linguagens de programação, facilitando o acoplamento com outro(s) modelo(s), dentre outros.

6. CONCLUSÃO

Dois *middleware* foram desenvolvidos no decorrer deste trabalho. O primeiro *middleware* permite a integração de modelos desenvolvidos em Fortran com outras linguagens de programação, permitindo a criação de um pacote (RSimpleCrop) integrando o R com um modelo Fortran, removendo todas as dependências de arquivos texto.

Um segundo *middleware* permitiu a extensão do DSSAT-CSM em C++, realizando todo o gerenciamento de entrada e saída dos dados, utilizando funções de leituras genéricas configuradas em tempo de execução, permitindo que os usuários formatem os arquivos de saídas sem a necessidade de alterar código fonte. Além disto, permitiu o uso de novos formatos de arquivos, fácil integração com sistemas de informação e facilitar acoplamento com outros modelos.

Neste contexto, conclui-se que o objetivo do trabalho foi atingido, pois os *middlewares* foram desenvolvidos e compilados com tecnologias de código fonte aberto, permitindo assim integração e evoluções de modelos desenvolvidos em Fortran aumentando as suas vidas uteis.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] CHWIF, L.; MEDINA, A. C. *Modelagem e simulação de eventos discretos*. [S.l.]: Afonso C. Medina, 2006.

[2] PEDRINI, J. E. *Acomplando um modelo de doenças ao modelo Cropgro-Soybean: Ferrigem Asiática da Soja*. 128 p. Dissertação (Mestrado) — Universidade de Passo Fundo, Passo Fundo, 2010.

[3] DSSAT Foundation. 2016. Disponível em: <https://http://dssat.net/about>. Acesso em: Dez. 10, 2016.

[4] JR, T. F. B. *Support for model coupling: an interface-based approach*. Tese (Doutorado) — University of Oregon, 2006.

[5] PAVAN, W. Técnicas de engenharia de software aplicadas à modelagem e simulação de doenças de plantas. *Universidade de Passo Fundo*, 2007.

[6] RECH, G.; PAVAN, W.; HöLBIG, C. A. Modelo de simulador aplicado ao desenvolvimento de culturas. In: ANAIS DO SIMPÓSIO DE COMPUTAÇãO APLICADA. PASSO FUNDO : UNIVERSIDADE DE PASSO FUNDO, 2010. Passo Fundo, 2010.

[7] R CoreTeam. *Writing R Extensions*. Vienna: R Foundation for Statistical Computing, 2015. 31 p.

[8] ALMEIDA, C. A. de. Criação de um pacote para visualização gráfica de distribuições densidade de probabilidade utilizando o software r. Curitiba, p. 178, 2006.

[9] JONES, J. et al. The DSSAT cropping system model. *European Journal of Agronomy*, v. 18, n. 3-4, p. 235–265, jan 2003.

[10] RICHARDSON, C. W. Stochastic simulation of daily precipitation, temperature, and solar radiation. *Water Resources Research*, Wiley Online Library, v. 17, n. 1, p. 182–190, 2 1981. ISSN 1944-7973.

[11] RICHARDSON, C. Weather simulation for crop management models. *Transactions of the ASAE*, American Society of Agricultural and Biological Engineers, v. 28, n. 5, p. 1602–1606, 1985.

[12] GENG, S.; VRIES, F. W. P. de; SUPIT, I. A simple method for generating daily rainfall data. *Agricultural and Forest Meteorology*, Elsevier, v. 36, n. 4, p. 363–376, 1986.

[13] GENG, S. et al. A program to simulate meteorological variables: Documentation for simmeteo, agronomy progress rep 204. *Department of Agronomy and Range Science, University of California, Davis, California*, 1988.

[14] RITCHIE, J.; OTTER, S. Description and performance of ceres-wheat: a user-oriented wheat yield model. *ARS-United States Department of Agriculture, Agricultural Research Service (USA)*, 1985.

[15] JONES, J.; RITCHIE, J. Crop growth models. *IN: Management of Farm Irrigation Systems. American Society of Agricultural Engineers, St. Joseph, MI. 1990. p 63-89, 10 fig, 2 tab, 72 ref.*, 1990.

[16] JONES, J. Decision support systems for agricultural development. In: *Systems approaches for agricultural development*. [S.l.]: Springer, 1993. p. 459–471.

[17] RITCHIE, J. Soil water balance and plant water stress. In: *Understanding options for agricultural production*. [S.l.]: Springer, 1998. p. 41–54.

[18] SERVICE, U. S. S. C. Scs national engineering handbook, section 4: hydrology. In: ____. [S.l.: s.n.], 1972. cap. 4,10.

[19] WILLIAMS, J.; JONES, C.; DYKE, P. A modeling approach to determining the relationship between erosion and soil productivity. *Transactions of the ASAE*, American Society of Agricultural and Biological Engineers, v. 27, n. 1, p. 129–0144, 1984.

[20] GODWIN, D. et al. Nitrogen dynamics in soil-plant systems. *Modeling plant and soil systems.*, American Society of Agronomy, Inc., p. 287–321, 1991.

[21] GODWIN, D.; SINGH, U. Nitrogen balance and crop response to nitrogen in upland and lowland cropping systems. In: *Understanding options for agricultural production*. [S.l.]: Springer, 1998. p. 55–77.

[22] SELIGMAN, N.; KEULEN, H. v. Papran: A simulation model of annual pasture production limited by rainfall and nitrogen. In: WAGENINGEN, NETHERLANDS, CENTRE FOR AGRICULTURAL PUBLISHING AND DOCUMENTATION, 1981. *Simulation of nitrogen behaviour of soil-plant systems: papers of a workshop, Models for the behaviour of nitrogen in soil and uptake by plant, comparison between different approaches, Wageningen, the Netherlands, January 28-February 1, 1980/editors MJ Frissel and JA van Veen*. [S.l.], 1981.

[23] GIJSMAN, A. J. et al. Modifying dssat crop models for low-input agricultural systems using a soil organic matter–residue module from century. *Agronomy Journal*, American Society of Agronomy, v. 94, n. 3, p. 462–474, 2002.

[24] PARTON, W. J.; STEWART, J. W.; COLE, C. V. Dynamics of c, n, p and s in grassland soils: a model. *Biogeochemistry*, Springer, v. 5, n. 1, p. 109–131, 1988.

[25] PARTON, W. J. et al. A general model for soil organic matter dynamics: sensitivity to litter chemistry, texture and management. In: SOIL SCIENCE SOCIETY OF AMERICA INC. *Quantitative modeling of soil forming processes: proceedings of a symposium sponsored by Divisions S-5 and S-9 of the Soil Science Society of America in Minneapolis, Minnesota, USA, 2 Nov. 1992.* [S.l.], 1994. p. 147–167.

[26] RITCHIE, J. T. Model for predicting evaporation from a row crop with incomplete cover. *Water resources research*, Wiley Online Library, v. 8, n. 5, p. 1204–1213, 1972.

[27] DOORENBOS, J. *Guidelines for predicting crop water requirements*. [S.l.], 1977.

[28] BOOTE, K. J.; JONES, J. W.; HOOGENBOOM, G. Simulation of crop growth: Cropgro model. Marcel Dekker, 1998.

[29] CAMARGO, M. B. P. de; BRUNINI, O.; MIRANDA, M. A. C. de. Temperatura-base para cálculo dos graus-dia para cultivares de soja em são paulo. *Pesquisa Agropecuária Brasileira*, v. 22, n. 2, p. 115–121, 1987.

[30] HUNT, L.; WHITE, J.; HOOGENBOOM, G. Agronomic data: advances in documentation and protocols for exchange and use. *Agricultural Systems*, Elsevier, v. 70, n. 2, p. 477–492, 2001.

[31] BATCHELOR, W. et al. Extending the use of crop models to study pest damage. *Transactions of the ASAE*, American Society of Agricultural and Biological Engineers, v. 36, n. 2, p. 551–558, 1993.

[32] REYNOLDS, J. F.; ACOCK, B. Modularity and genericness in plant and ecosystem models. *Ecological Modelling*, Elsevier, v. 94, n. 1, p. 7–16, 1997.

[33] ACOCK, B.; REYNOLDS, J. F. The rationale for adopting a modular generic structure for crop simulators. In: *International Symposium on Models for Plant Growth, Environmental Control and Farm Management in Protected Cultivation 248*. [S.l.: s.n.], 1988. p. 391–400.

[34] PORTER, C. H.; BRAGA, R.; JONES, J. W. An Approach for Modular Crop Model Development. n. 99, 1999.

[35] HÖLBIG C. A. ; PAVAN, W. . M. A. . F. J. M. C. . Z. T. Rcropgro package: an approach for encapsulating fortran coded models using r language. In: 11TH INTERNATIONAL CONFERENCE APPLIED COMPUTING 2014. Porto, 2014. p. 119–126.

[36] TOEBE, J. *UM MODELO BASEADO EM AGENTES PARA O CICLO DE VIDA DE AFÍDEOS: APLICAÇÃO NA INTERAÇÃO AFÍDEO-PLANTA-VIRUS*. Tese (Doutorado) — Universidade de Passo Fundo, Passo Fundo, 2006.

[37] BERGEZ, J.-E. et al. An open platform to build, evaluate and simulate integrated models of farming and agro-ecosystems. *Environmental Modelling & Software*, 2013. ISSN 13648152.

[38] BROWN, H. E. et al. Plant Modelling Framework: Software for building and running crop models on the APSIM platform. *Environmental Modelling & Software*, v. 62, p. 385–398, Dec 2014.

[39] WU, Y.; LIU, S.; YAN, W. A universal Model-R Coupler to facilitate the use of R functions for model calibration and analysis. *Environmental Modelling & Software*, v. 62, p. 65–69, Dec 2014.

[40] EWERT, F. et al. Crop modelling for integrated assessment of risk to food production from climate change. *Environmental Modelling & Software*, 2014.

[41] A coupled model of water, heat and mass transfer using object orientation to improve flexibility and functionality. *Environmental Modelling & Software*, v. 16, n. 1, p. 37–46, 2001.

[42] JONES, J. W.; KEATING, B. a.; PORTER, C. H. Approaches to modular model development. *Agricultural Systems*, v. 70, n. 2-3, p. 421–443, 2001.

[43] THYER, M. The open source rfortran library for accessing r from fortran, with applications in environmental modelling. *Environmental Modelling & Software 26 (2011) 219e234*, v. 26, 2011.

[44] R Development Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2010. ISBN 3-900051-07-0. Disponível em: <http://www.R-project.org>.

[45] FORTRAN. *Mixed-Language Programming interoperability with C*. 2015. Disponível em: <https://gcc.gnu.org/onlinedocs/gfortran/ Interoperability-with-C.html>. Acesso em: Jan. 14, 2017.

[46] R Development Core Team. *Writing R Extensions*. [S.l.], 2016. Disponível em: <https://cran.r-project.org/doc/manuals/r-release/R-exts.htmlInterface-functions-$_0 02eC - and -_0 02eFortran$>.

[47] COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. [S.l.]: pearson education, 2005.

[48] IFORT. *Intel Fortran Compiler*. 2017. Disponível em: <https://software.intel.com/en-us/fortran-compilers>. Acesso em: Jan. 14, 2017.

[49] GFORTRAN. *GNU Fortran project*. 2017. Disponível em: <https://gcc.gnu.org/fortran/>. Acesso em: Jan. 14, 2017.

[50] GCC. *GNU Compiler Collection*. 2015. Disponível em: <https://gcc.gnu.org/>. Acesso em: Jan. 14, 2017.

[51] PAVAN, W. *Técnicas de Engenharia de Software Aplicadasa Modelagem e Simulaçao de Doenças de Plantas*. Tese (Doutorado) — Universidade de Passo Fundo, 2007.