CRICTE 2017
XXVIII Congresso Regional de Iniciação Científica e Tecnológica em Engenharia
UNIJUÍ UNIVERSIDADE REGIONAL

GERAÇÃO AUTOMÁTICA DE MODELOS DE SIMULAÇÃO PARA SOLUÇÕES DE INTEGRAÇÃO

Igor G. Haugg
Acadêmico do Programa de Pós-Graduação em Modelagem Matemática
Universidade Regional do Noroeste do Estado do RS (UNIJUÍ)
igor.haugg@unijui.edu.br

Rafael Z. Frantz
Professor do Programa de Pós-Graduação em Modelagem Matemática
Universidade Regional do Noroeste do Estado do RS (UNIJUÍ)
rzfrantz@unijui.edu.br

Resumo. *Integrar aplicações é fazer com que aplicações isoladas, possam colaborar e dar suporte a novos processos de negócio. Soluções de integração podem ser desenvolvidas através de diferentes plataformas, as quais fornecem ferramentas para modelar e implementar as soluções, tais como, linguagem de domínio específico, kit de ferramentas de desenvolvimento, ferramenta de monitoramento e motor de execução. O desempenho de uma solução de integração é dependente do desempenho do motor de execução. Para aumentar seu desempenho, os motores trabalham com um conjunto de threads, que permitem a execução paralela da solução de integração. A configuração do número de threads depende de cada solução de integração e é definida a partir do conhecimento empírico dos engenheiros de software. Esta abordagem traz riscos, pois o dimensionamento incorreto do número de threads pode prejudicar o desempenho dos motores de execução. Este artigo apresenta uma proposta de algoritmo para auxiliar no dimensionamento correto de threads.*

Palavras-chave: *Integração de aplicações empresariais. Modelos de simulação. Geração automática de modelos.*

1. INTRODUÇÃO

As empresas normalmente possuem diversas aplicações que realizam o suporte aos seus processos de negócio. O ecossistema de software das empresas é normalmente formado por aplicações desenvolvidas internamente ou adquiridas de terceiros. Este ecossistema muitas vezes torna-se heterogêneo, com aplicações que foram desenvolvidas com diferentes linguagens de programação, sistemas operacionais, e, geralmente, não foram projetadas para trabalhar de forma colaborativa [6].

Integração de aplicações empresariais (EAI) [4] é um campo de pesquisa que oferece metodologias, técnicas e ferramentas.

Existem diversas plataformas disponíveis para o desenvolvimento de soluções de integração, dentre as *open-source*, destacam-se Camel [1], Spring Integration [5], Mule[2], e Guaraná[6]. Essas plataformas suportam os padrões de integração documentados por Hohpe e Woolf [4], que se tornaram referência na comunidade EAI. Normalmente, as plataformas de integração oferecem uma linguagem de domínio específico, um kit de ferramentas de desenvolvimento, uma

ferramenta de monitoramento e um motor de execução. Linguagem de domínio específico serve para projetar modelos conceituais. O kit de desenvolvimento é um conjunto de ferramentas para transformar o modelo conceitual em código executável. A ferramenta de monitoramento é utilizada para acompanhar, em tempo de execução, o funcionamento da solução de integração e detectar erros que possam ocorrer durante sua execução. O motor de execução proporciona o suporte necessário para a execução dessas soluções de integração. Por essa razão, o desempenho dos motores de execução está relacionado com o desempenho das soluções de integração. A arquitetura dos motores de execução das plataformas estudadas está organizada em uma fila, a qual é utilizada para anotar as tarefas da solução de integração que podem ser executadas, a execução destas tarefas é realizada por um conjunto de *threads* configuradas no motor de execução.

A configuração do motor de execução é realizada através da definição do número de *threads* que devem ser utilizadas. Essa configuração é importante para garantir um bom desempenho e também atingir os requisitos de qualidade de serviço que o usuário deseja, tal como, o número mínimo de mensagens que a solução deve processar por unidade de tempo. Atualmente, a configuração do número de *threads* é definida a partir do conhecimento empírico dos engenheiros de software. Esta abordagem traz consigo riscos, pois o dimensionamento incorreto do número de *threads* tem impacto no desempenho. Um baixo número de *threads* pode levar ao acúmulo de tarefas, enquanto um número muito elevado e desnecessário de *threads* representa um desperdício de recursos computacionais, e, também pode causar um acúmulo de tarefas na fila, já que o tempo para o gerenciamento das *threads* gasto pelo sistema operacional aumenta. As soluções de integração são representadas em arquivos XML.

Este artigo propõe um algoritmo para realizar a geração automática de modelos de simulação tendo como base os modelos XML desenvolvidos na plataforma de integração Guaraná.

O restante deste artigo tem sua estrutura organizada como segue. A Seção 2 apresenta o processo para geração de modelos de simulação. A Seção 3 apresenta o algoritmo para a geração dos modelos. A Seção 4 proporciona um exemplo de saída gerada pelo algoritmo. Finalmente, a Seção 5 discute as principais conclusões.

2. GERAÇÃO DO MODELO

O processo para a geração do modelo de simulação toma como entrada o modelo da solução de integração e dados sobre a configuração da execução, conforme Fig. 1. O modelo da solução é desenvolvido com o editor da plataforma Guaraná. Nela, é possível modelar graficamente o fluxo da solução de integração e realizar a exportação deste modelo, para o formato XML.

Os dados de execução são informações que o usuário precisa informar ao algoritmo, tais como: tempo de simulação, taxa de entrada, número de threads e tempo de execução de cada tarefa. Tempo de simulação é onde os usuários poderão informar o tempo total de execução para o modelo de simulação gerado. A taxa de entrada representa quantas mensagens a solução de integração recebe a cada unidade de tempo. Número de threads determina com quantas threads o modelo de simulação fará os experimentos, o tempo de execução de cada tarefa representa o tempo total para executar individualmente uma tarefa da solução de integração, e é calculado de forma automática pelo algoritmo.

Com base em um template o modelo da solução é transformado ao formato *Model Definition Language* (MDL), para integrar o modelo de simulação, o qual pode ser utilizado pelo software de simulação Matlab Simulink. O motor de execução também precisou ser modelado no formato MDL e

integra o modelo de simulação final. No modelo de simulação, a parte variável é composta pelo modelo da solução de integração e os dados de entrada.

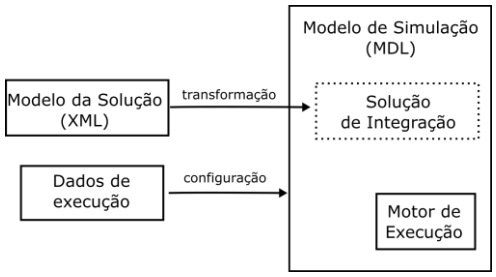

Figura 1. Processo de Geração.

3. ALGORITMO

Esta seção descreve o algoritmo proposto para a geração do modelo de simulação. A Figura 2 apresenta o pseudo-codigo deste algoritmo. Nas linhas 1, 2 e 3, realiza-se as leituras de arquivos que contém informações sobre o modelo da solução de integração, os tempos de execução das tarefas e os dados de execução. Na linha 5, a variável *model* recebe o modelo de simulação e na linha 6, a variável *tasks* recebe todas as tarefas. Na linha 8, a variável *times* recebe o tempo de execução de cada tarefa, enquanto que na linha 9, a variável *data* recebe os dados de execução informado pelo usuário. Após a leitura dos dados de entrada, e a criação de variáveis para receber estas informações, inicia-se as rotinas do algoritmo de geração. O código entre a linha 13 e 17, é repetido, para todas as tarefas lidas na variável *tasks*. Na linha 13, adiciona-se o tempo de serviço recebido na variável *times*, para a tarefa atual. Na linha 14, adiciona-se a taxa de entrada, a partir do retorno da função *calcularTaxaDeEntrada* que calcula a taxa de entrada. Nesta função, é calculada a taxa de entrada a partir da taxa informada pelo usuário. Na linha 15, insere-se um tipo para a tarefa, tendo como funcionalidade, diferenciar uma tarefa da outra dentro do modelo de simulação. Na linha 16, adiciona-se para a tarefa atual, a próxima tarefa presente no fluxo de integração. Essa informação é recebida através da execução da função *descobrirProximaTarefaDoFluxo*. Na linha 17, incrementa-se a variável *i*, para que na próxima execução seja analisada a próxima tarefa. A seguir, na linha 20, cria-se o modelo de simulação, na linha 21, o modelo de simulação recebe o número de *threads* informado pelo usuário, na linha 22, o modelo de simulação recebe o tempo de simulação informado pelo usuário e na linha 23 adiciona-se no modelo de simulação as tarefas com todas as suas configurações. Na sequência, na linha 25, cria-se a variável *runtimeFN* para receber o modelo do motor de execução, na linha 26, a variável *read* recebe o modelo do motor. Por fim, na linha 27, o modelo de simulação recebe o modelo do motor de execução.

```
1:  modelFN <- arquivo com o modelo da solução de integração
2:  timesFN  <- arquivo com tempos de execução para tarefas
3:  dataFN   <- arquivo com os dados configurados pelo usuário
4:
5:  model <- readModel(modelFN)
6:  tasks  <- model.getTasks()
7:
8:  times <- getExecutionTimes(tasks,timesFN)
9:  data  <- readExecutionData(dataFN)
10:
11: i <- 0
12: while (i > tasks.size()) do
13:    tasks[i].setServiceTime( times[i].getServiceTime() )
14:    tasks[i].setEntryTax(calcularTaxaDeEntrada(model[i].getTaxa()))
15:    tasks[i].setType(i)
16:    tasks[i].setFindNext(descobrirProximaTarefaDoFluxo(model[i]))
17:    i <- i + 1;
18: end while
19:
20: smodel <- new SimulationModel()
21: smodel.setNumberOfThreads(data.getNumberOfThreads())
22: smodel.setSimulationTime(data.getNumberOfThreads())
23: smodel.addTasks(tasks)
24:
25: runtimeFN <- arquivo com o modelo do motor de execução
26: run <- readRuntimeModel(runtimeFN)
27: smodel.setRuntime(run)
```

Figura 2. Algoritmo para Geração.

4. SÁIDA DO ALGORITMO

Esta seção descreve a saída de uma das tarefas geradas pelo algoritmo. A Figura 3, apresenta a saída para a tarefa correlator. Em A, é possível visualizar a descrição da tarefa no formato XML, e em B, a descrição da mesma tarefa no formato MDL, após a geração pelo algoritmo. No formato XML, a

tarefa possui cardinalidade, um identificador, e além disso, é interligada a duas saídas, definidas como output[0] e output[1], que são responsáveis por enviar os dados para as próximas tarefas do fluxo de integração, que neste caso é a tarefa Context-based Enricher.

No formato MDL, são necessárias diversas configurações extras, para o bom funcionamento do modelo de simulação. Neste modelo, as tarefas passam a ser chamadas de blocos, cada bloco possui um tipo, chamado de EntityGenerator, capaz de gerar entidades no modelo de simulação, um nome, um identificador e uma quantidade de portas de saída. Também há uma série de instruções para a transformação gráfica, como: posição, ordem, modo de porta de saída, portas de entrada e saída. Além disso, a taxa de entrada é configurada a partir de um tempo entre gerações consecutivas, denominado Intergenartion Time Action.

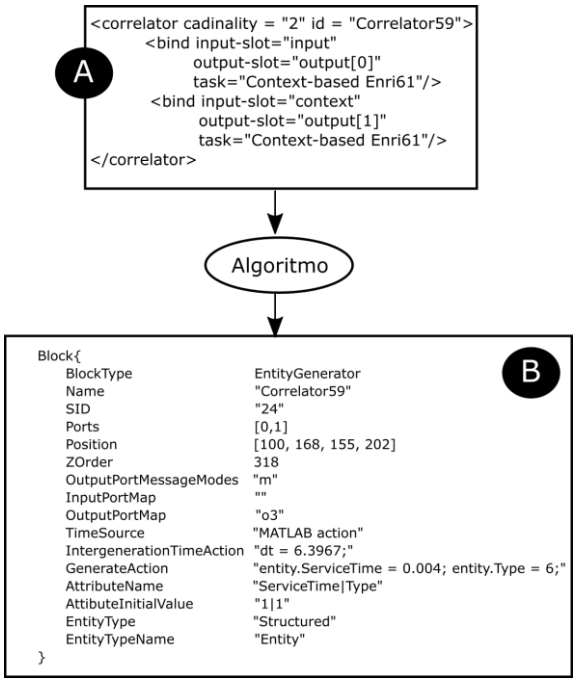

Figura 3. Saída do Algoritmo.

Agradecimentos

O trabalho do primeiro autor é suportado pelo CNPq. O segundo autor foi suportado parcialmente pela Capes, processo nº 88881.119518/2016-01 do Programa de Pós-doutorado no Exterior.

5. REFERÊNCIAS

[1] C. Ibsen, J. C. Praca, Camel in action, Manning Publications Co, 2010.

[2] D. Dossot, J. Emic, V. Romero, Mule in action, Manning Publications Co, 2014.

[3] D. S. Linthicum, Enterprise application integration, Addison-Wesley Professional, 2000.

[4] G. Hohpe, B. Woolf, Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional, 2004.

[5] M. Fisher, J. Partner, M. Bogoevici, I. Fuld, Spring integration in action, Manning Publications Co, 2012.

[6] R.Z. Frantz, R. Chorchuelo and F. Roos-Frantz, “On the design of a maintainable software development kit to implement integration solutions,” Journal of Systems and Software, v. 111, p. 89-104, 2016.

CONSIDERAÇÕES FINAIS

As plataformas de integração executam as soluções de integração através dos motores de execução. Estes motores são fundamentais para o desempenho das soluções, e buscar o número ideal de *threads* para a execução destes motores tem grande importância. Para descobrir o número ideal de *threads*, este artigo apresentou o desenvolvimento de um algoritmo para a geração de modelos de simulação de forma automática. O algoritmo realiza de forma rápida e dinâmica a geração de modelos formais aptos serem simulados.