Modalidade do trabalho: Relatório técnico-científico
Evento: XXIV Seminário de Iniciação Científica

FRAMEWORK DE COMPARAÇÃO DA EVOLUÇÃO DO APACHE CAMEL VISANDO A COMPLEXIDADE DE SEU GRAU DE MANUTENÇÃO[1]

Matheus H. Rehbein[2], Fabricia Roos-Frantz[3].

[1] Pesquisa de Iniciação Científica desenvolvida no Grupo de Pesquisa em Computação Aplicada (GCA) da UNIJUI.
[2] Bolsista PROBIC/FAPERGS, Ciência da Computação, matheus.rehbein@unijui.edu.br
[3] Professora Orientadora, frfrantz@unijui.edu.br

Introdução

Com a evolução de uma empresa, torna-se necessário a utilização de mais de uma aplicação em seu ecossistema de software (MESSERSCHMITT, SZYPESKI et al., 2005). A utilização de uma única aplicação que contenha todas as funcionalidades que uma empresa necessita pode se tornar custoso e portanto inviável (FRANTZ, QUINTERO, CORCHUELO, 2011), assim surge a necessidade de uma integração entre as aplicações. Para Hohpe e Woolf (2003), essa integração deve reutilizar as funcionalidades das aplicações, e quando surgir uma nova necessidade, implementá-la sem perder seu estado original.

Com a necessidade de integração entre aplicações, surgiu a necessidade de criar tecnologias para facilitar este processo. Segundo Hohpe e Woolf (2003) há quatro tipos de integração, são eles: Transferência de arquivo, Compartilhamento de banco de dados, Chamada de procedimento remoto, e Sistema de mensagem. Ferramentas como Apache Camel, Spring Integration, Mule e Guaraná, destacam-se entre as demais (FRANTZ, CORCHUELO, ROOS-FRANTZ, 2015). Essas ferramentas atendem a diversos padrões de integração descritos por Hohpe e Woolf (2003), além de possuírem sistemas de mensagem.

Sistema de mensagem é a forma com que uma aplicação se comunica com as demais. É utilizado mensagens para fazer a comunicação, o conteúdo delas é um determinado tipo de dado. Esse dado é enviado para a aplicação que o solicitou para que possa ser processado. É possível fazer com que uma aplicação receba uma mensagem sem estar funcionando no momento do envio, o que caracteriza o sistema de comunicação como assíncrono. Para garantir que uma mensagem chegue a seu destino utiliza-se canais. Caso a aplicação destinatária da mensagem não esteja em funcionamento no momento do envio da mensagem, esta ficará aguardando em uma fila destinada a esta mesma aplicação, a mensagem posterior só será processada após a execução da mensagem que está aguardando na fila. Portanto, este canal de comunicação entre as aplicações é representado por filas de mensagens (HOHPE, WOOLF, 2003).

Ferramentas de integração podem constantemente necessitar uma nova funcionalidade, correção de algum problema, ou melhora em sua performance. Essas etapas são conhecidas como Manutenção de Software (RADATZ, GERACI, KATKI, 1990). A partir de uma extração de dados é possível a realização de uma medição para definir o grau de manutenibilidade de uma ferramenta. Quanto

maior for o grau de manutenibilidade, mais complicado será adicionar, aprimorar ou manter uma funcionalidade do código.

Para este trabalho, optou-se pela utilização da ferramenta Apache Camel, que possui código open-source podendo assim ser feita a extração de informações. Apache Camel possui um sistema de mensagens assíncronas, para a comunicação entre aplicações. Segundo Ibsen e Anstey (2010), outra característica fundamental é que o Camel não se restringe a um determinado tipo de dado, ou seja, cada aplicação pode ter seu próprio pedrão de dados, isso possibilitando a integração entre aplicações sem necessitar a conversão da mensagem para um único tipo de dados.

Este trabalho tem como objetivo medir o grau de manutenibilidade da ferramenta de integração Apache Camel, para então predizer como será seu comportamento durante a manutenção.

Metodologia

Após a leitura de artigos científicos para compreensão do problema, foi realizado um levantamento de todas as versões disponíveis do Camel e destas versões foram selecionadas dezesseis. Em seguida utilizou-se a ferramenta Métrics para realizar a medição. Esta ferramenta permite a extração de dados do software analisado para posterior análise. Esses dados permitem identificar como a manutenção do software está evoluindo, ou seja, se está se mantendo estável ou aumentando sua complexidade.

Após realizada a extração dos dados, foram obtidas diversas métricas das quais foram selecionadas duas: McCabe cyclomatic complexity (MCC) e Number of lines in methods (MLC).

A medida MCC é utilizada para definir o quão complexo é um método, segundo McCabe (1976) este valor não deve exceder dez, pois com o aumento deste valor, a manutenção deste método se tornará mais difícil. Já a medida MLC define a quantidade de linhas que um método possui. Henderson-Sellers (1995) afirma que este valor não pode ser maior que cinquenta. Alem disso, quanto maior for este valor, mais difícil será de manter e entender o método (FRANTZ, CORCHUELO, ROOS-FRANTZ, 2015).
Análise dos Resultados

Versões	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10	2.11	2.12	2.13	2.14	2.15	2.16
MCC - Mean	1,6	1,61	1,6	1,63	1,66	1,67	1,68	1,67	1,64	1,65	1,67	1,69	1,69	1,7	1,71	1,7	1,69
MCC - Max	42	42	43	46	46	46	46	46	46	46	48	48	48	50	50	50	64
MLC - Mean	4,16	4,14	4,2	4,3	4,41	4,46	4,54	4,52	4,57	4,62	4,71	4,73	4,71	4,75	4,79	4,79	4,73
MLC - Max	103	103	103	113	112	137	141	141	150	153	163	120	122	128	131	135	150

Figura 1: Framework de Comparação.

Como ressaltado anteriormente, o valor referente a MCC não deve ultrapassar dez. Na Figura 1 é apresentado o valor médio e o valor máximo do MCC. O valor médio manteve-se com pequenas

variações nas distintas versões, não muito significativas, porém o valor máximo mostrou-se com crescimento. Pode-se observar que a primeira versão analisada do Camel possuiu método com o valor até 42, mais de quatro vezes maior que o recomendado. Este valor foi aumentando até 50 na versão 2.15, e então ocorreu uma variação, fazendo com que chegasse a 64 na versão 2.16. Essa variação ocorreu devido a reestruturação do método "getManagedObjectForProcessor" pertencente a classe "DefaultManagementObjectStrategy" presente no package "org.apache.camel.management". Já o valor médio desta classe encontra-se com 6,83, mesmo possuindo o método com o maior valor de MCC. Sendo que esta classe possui 12 métodos, para que aconteça uma variação deste tipo, é necessário que o valor dos outros métodos contidos na classe sejam próximos a 1. Também foi verificado que todos métodos que possuíram o valor máximo na versão tiveram o maior nível de MLC de seu package, acima do recomendável.

Também na Figura 1, apresenta-se os valores referentes a MLC, sendo eles o valor máximo e a média. Como citado anteriormente, é recomendado que um método não possua mais de cinquenta linhas de código. A média teve uma tendência de crescimento, porém nunca atingiu a quantidade de 5 linhas. Já o valor máximo teve crescimento até a versão 2.10, alcançando o valor de 163 linhas de códigos. O método que possuiu este valor não se encontra na versão seguinte, ele também era responsável pelo maior valor MCC presente. Na versão seguinte, o número máximo caiu para 120, mas mesmo assim continuou crescendo até chegar ao valor 150, na última versão. O valor máximo da última versão é também o método responsável pelo maior MCC, "getManagedObjectForProcessor". Mesmo o método ficando com um valor elevado, o valor médio da métrica MLC na classe que o método pertence esteve abaixo do recomendado, possuindo 18,17. O package da classe que possui o método ficou com a média em 6,63, muito próximo da média geral.

Durante a evolução das versões o valor máximo da medida MCC nunca esteve dentro do recomendado, possuindo versão com até seis vezes maior, isso irá resultar em uma manutenção mais complexa. Também identificou-se que os valores referentes ao número máximo de linhas de códigos nos métodos esteve maior que o recomendável, tornando assim mais complicado a manutenção e entendimento deste método. Mesmo tendo uma queda na versão 2.11, ele continuou crescendo e atingiu um valor três vezes maior que o recomendado na última versão. Os valores que representam as médias, tanto do MCC como do MLC, estiveram bastante estáveis e baixos, isso ocorre porque grande parte dos métodos possuem valores abaixo do recomendado. Portando, pode-se afirmar que não são todos os métodos que estão com um grau de manutenibilidade alto. Porém, com os valores máximos obtidos a partir do MCC e MLC, provou-se que algumas partes do Camel possuem um alto grau de manutenção, dificultando sua evolução.

Conclusão

Após análise de distintas versões, ficou visível que a evolução do Apache Camel teve um impacto em seu grau de manutenibilidade referente a complexidade de alguns métodos, tornando a manutenção mais complexa. Tornou-se visível que os métodos que possuem o valor máximo da

métrica MCC alto, também possuem um nível de MLC maior que o recomendado, podendo-se afirmar que um método com MLC alto, poderá influenciar na medida MCC do mesmo método.

A partir deste trabalho, é possível afirmar que o grau de manutenibilidade da ferramenta Camel vem crescendo e tornando-se cada vez mais complexo. Para diminuir esses efeitos, deve-se reestruturar diversas partes do código. O processo de reestruturação pode se tornar demorado ou até mesmo inviável, podendo assim acarretar em uma ferramenta com um tempo de manutenção possivelmente alto.

Palavras-Chave: Integração de Aplicações; Manutenção de Software; Código Aberto.

Agradecimentos

Agradeço a Fapergs pela concessão da bolsa de Iniciação Científica (PROBIC) que permitiu o desenvolvimento desta pesquisa. Também agradeço a todos os membros do grupo de pesquisa GCA por todo conhecimento compartilhado.

Referências Bibliográficas

FRANTZ, Rafael Z.; CORCHUELO, Rafael; ROOS-FRANTZ, Fabricia: A methodology to evaluate the maintainability of enterprise application integration frameworks. International Journal of Web Engineering and Technology. Inderscience Publishers (IEL), 2015, p. 334-354.

FRANTZ, Rafael Z.; QUINTERO, Antonio M. Reina; CORCHUELO, Rafael: A domain-specific language to design enterprise application integration solutions. International Journal of Cooperative Information Systems. World Scientific, 2011, p. 143-176.

HENDERSON-SELLERS, Brian: Object-oriented metrics: measures of complexity. Prentice-Hall, Inc., 1995.

HOHPE, Gregor; WOOLF, Bobby: Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions. The Addison Wesley Signature Series, Boston, 2003, p. 12-75.

IBSEN, Claus; ANSTEY, Jonathan: Camel in action. Manning Publications Co., 2010.

MCCABE, Thomas J.: A complexity measure. Software Engineering, IEEE Transactions on, 1976, p. 308-320.

MESSERSCHMITT, David G.; SZYPESKI, Clemens et al. Software ecosystem: understanding an indispensable technology and industry. MIT Press Books. The MIT Press, 2005.

RADATZ, Jane; GERACI, Anne; KATKI, Freny: IEEE standard glossary of software engineering terminology. IEEE Std,1990, p. 1-3.

Modalidade do trabalho: Relatório técnico-científico
Evento: XXIV Seminário de Iniciação Científica