ANAIS PROCEEDINGS

CBSOFT 2015

BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE

BELO HORIZONTE

WDES 2015

9th WORKSHOP ON DISTRIBUTED SOFTWARE DEVELOPMENT, SOFTWARE ECOSYSTEMS AND SYSTEMS-OF-SYSTEMS

CBSOFT.ORG

WDES 2015

9thWORKSHOP ON DISTRIBUTED SOFTWARE DEVELOPMENT, SOFTWARE ECOSYSTEMS AND SYSTEMS-OF-SYSTEMS

September 23rd, 2015
Belo Horizonte – MG, Brazil

VOLUME 01
ISSN: 2178-6097

ANAIS | *PROCEEDINGS*

APRESENTAÇÃO

O Workshop em Desenvolvimento Distribuído de Software, Ecossistemas de Software e Sistemas-de-Sistemas (WDES 2015), em sua nona edição, traz como principal tema os "Impactos de Ecossistemas de Software sobre a Qualidade no Desenvolvimento Distribuído de Software e em Sistemas-de-Sistemas". O WDES constitui um fórum para apresentação e discussão de resultados e experiências de pesquisadores e praticantes das áreas de Desenvolvimento Distribuído de Software (DDS), Ecossistemas de Software (ECOS) e Sistemas-de-Sistemas (SoS), visando a geração de conhecimento que possa viabilizar projetos de sucesso nessas três áreas e/ou nas suas relações. Além disso, o workshop visa ampliar as possibilidades de colaboração em âmbito nacional e internacional, bem como consolidar as pesquisas em DDS, ECOS e SoS como uma área estratégica em Engenharia de Software no Brasil, como ocorre no exterior.

O Comitê Diretivo do WDES 2015 é constituído por três pesquisadores de cada uma das áreas envolvidas no workshop. Por sua vez, o Comitê de Programa do WDES 2015 é formado por 28 pesquisadores de instituições do Brasil (19) e do exterior (9), com atuação e produção relevantes nas áreas de pesquisa envolvidas no workshop, além de ter 24 revisores externos. Os membros do Comitê de Programa conduziram um processo rigoroso de revisão, sendo que cada artigo foi avaliado por pelo menos três membros.

O WDES 2015 recebeu número recorde de submissões desde a primeira edição em 2007. Foram submetidos 32 artigos (17 em inglês e 15 em português). Após o processo de revisão e consenso do Comitê de Programa, além da análise final do Comitê Diretivo, foram aceitos 9 artigos completos (28%) e 5 artigos curtos (16%). Haverá premiação do melhor artigo e convite para submissão de versão estendida para uma Edição Especial dos Workshops do CBSoft 2015. Para estimular a discussão de pesquisas com potencial, 4 trabalhos serão ainda apresentados como pôsteres. Por fim, o WDES vai contar com um painel com especialistas nas áreas do workshop e com uma dinâmica para a organização da Agenda de Pesquisa e de Colaboração do WDES.

É com satisfação que damos as boas-vindas aos autores e apresentadores de artigos, da academia e da indústria. Também recebemos com grande prazer os demais participantes do CBSoft 2015, que gostaríamos de convidar a tomar parte ativamente das discussões e momentos de integração proporcionados pelo workshop. Adicionalmente, gostaríamos de agradecer a todos os demais autores que submeteram seus artigos, aos membros do Comitê de Programa e do Comitê Diretivo, e aos organizadores e patrocinadores do CBSoft 2015 pelo suporte na realização deste workshop.

Esta edição é organizada conjuntamente pela Universidade Federal do Rio de Janeiro (COPPE/UFRJ), Universidade Federal de Lavras (UFLA) e IRISA – European University of Brittany / UBS. O evento é realizado na PUC Minas, campus Coração Eucarístico, em Belo Horizonte, Minas Gerais, no dia 23 de setembro, em conjunto com o VI Congresso Brasileiro de Software: Teoria e Prática (CBSoft 2015).

Esperamos que tenham uma ótima estada em Belo Horizonte!

Belo Horizonte, Setembro de 2015.

Rodrigo Santos (COPPE/UFRJ)
Heitor Costa (UFLA)
Flavio Oquendo (IRISA-UBS)

Coordenadores do Comitê de Programa do WDES 2015

FOREWORD

The Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES 2015), in its ninth edition, aims at putting together competencies and technologies of these three related areas: Distributed Software Development (DSD), Software Ecosystems (SECO), and Systems-of-Systems (SoS). This year's main topic is "Impacts of Software Ecosystems on the Quality in Distributed Software Development and Systems-of-Systems". This workshop consists of a forum for presentation and discussion of results and experiences of researchers and practitioners of DSD, SECO, and SoS. As the main goal, it intends to generate knowledge that makes it possible to conduct successful projects in theseareas. Besides, the workshop tries to leverage opportunities for national (and international) collaborations, as well as strengthen researches on DSD, SECO, and SoS as an strategic area in the Brazilian Software Engineering, as abroad.

The WDES 2015 Steering Committee is composed of three researchers from each workshop area.The WDES Program Committee is composed of 28 researchers from national (19) and international (9) institutions, and also 24 additional reviewers. Each submission was reviewed by at least three members of the program committee. All papers were judged on the basis of their clarity, relevance, originality, and contribution. On-line discussion on paper reviewsalso helped to solve divergent views regarding the evaluation of all papers. The Steering Committee then analyzed the ranking and scores of all submissions and endorsed the list of accepted papers.

In this WDES edition, we received 32papers (17 English papers and 15 Portuguese papers), reaching a record number of submissions since the first edition in 2007. Based on the reviews, 9full papers (28%) and 5 short papers (16%) were selected for presentation and publication on the workshop proceedings. Best paper will be invited to submit an extended version to a Special Issue of the Workshops of CBSoft2015.Additionally, 4 papers will be presented as posters since they discuss potential research ideas. Finally, we will promote a panel with experts in the workshop areas and a Special Session to define the WDES Research and Collaboration Roadmap.

We would like to thank the authors of submissions to WDES 2015. We would also like to thank the Program Committee members and the additional reviewersfor their time and dedication in reviewing and discussing papers submitted to the workshop.We thank the Steering Committee members for their support to this work. Finally, we would like to give special thanks to the general chairs of CBSoft 2015 for their effort to make this important event possible.

This WDES edition is organized by Federal University of Rio de Janeiro (COPPE/UFRJ), Federal University of Lavras (UFLA) and IRISA – European University of Brittany / UBS. This workshop will be held at PUC Minas, campus Coração Eucarístico, in Belo Horizonte, State of Minas Gerais, Brazil, on September 23rd, co-located with VI Brazilian Congress on Software: Theory and Practice (CBSoft 2015).

We welcome all the WDES 2015 attendees and wish a pleasant stay in Belo Horizonte.

Belo Horizonte, September 2015.

Rodrigo Santos (COPPE/UFRJ)
Heitor Costa (UFLA)
Flavio Oquendo (IRISA-UBS)

WDES 2015 PC Chairs

COMITÊ DE ORGANIZAÇÃO | *ORGANIZING COMMITTEE*

COMITÊ TÉCNICO | *TECHNICAL COMMITTEE*

COORDENADORES DO COMITÊ DE PROGRAMA | *PC CHAIRS*

Rodrigo Santos (COPPE/UFRJ)
Heitor Costa (UFLA)
Flavio Oquendo (IRISA – UBS)

COMITÊ DIRETIVO | *STEERING COMMITTEE*

Carina Alves (UFPE)
Cláudia Werner (COPPE/UFRJ)
Elisa Huzita (UEM)
Elisa Nakagawa (ICMC/USP)
Flavio Oquendo (IRISA – UBS)
Heitor Costa(UFLA)
José Carlos Maldonado (ICMC/USP)
Rodrigo Santos (COPPE/UFRJ)
Sabrina Marczak (PUCRS)

COMITÊ DE PROGRAMA | *PROGRAM COMMITTEE*

Alexandre L'Erario (UTFPR)
Aline Vasconcelos (IFF)
Arilo Claudio Dias-Neto (UFAM)
Carina Alves (UFPE)
Cláudia Werner (COPPE/UFRJ)
Daniela Cruzes (SINTEF)
Davi Viana (UFAM)
Elisa Huzita (UEM)
Elisa Nakagawa (ICMC/USP)
Fabio Silva (UFPE)
Guilherme Travassos (COPPE/UFRJ)
Igor Steinmacher (UTFPR)
Ivaldir de Farias Junior (Softex Recife)
Jan Bosch (Chalmers University of Technology)
Jennifer Benedí Pérez (Technical University of Madrid)
John Mcgrego r(Clemson University)
Josiane Kroll (University of Manitoba)
Khalil Drira (LAAS-CNRS)
Leonardo Murta (UFF)
Marco Gerosa (IME/USP)
Paris Avgeriou (University of Groningen)
Rafael Capilla (King Juan Carlos University)
Rafael Prikladnicki (PUCRS)

Ryan Azevedo (UFRPE)
Sabrina Marczak (PUCRS)
Simone Vasconcelos (IFF)
SlingerJansen (Utrecht University)
Tayana Conte (UFAM)

REVISORES EXTERNOS | *EXTERNAL REVIEWERS*

Adriana Lopes	Filipe Roseiro Côgo	Jessica Díaz
Anna Beatriz Marques	Frank José Affonso	Joyce Aline Oliveira
AwdrenFontão	George Valença	Lina Garcés
Catarina Costa	Gislaine Leal	Lucas B. Ruas de Oliveira
Cleyton Rodrigues	Igor ScalianteWiese	Marcelo França
Danilo Ribeiro	Ilhem Khlif	Maurício Aniche
Denis Zaniro	Isabella R. M. dos Santos	Rodrigo Lopes
Elisa Sattyam Cardozo	Ismael B. Rodriguez	Thaiana Lima

WEBSITE AND SUPPORT

Ivaldir de Farias Junior (Softex Recife)
Thaiana Lima (COPPE/UFRJ)
Luisa Hernández (UFLA)
Lucas B. Ruas de Oliveira (ICMC/USP)
Marcelo Benites Gonçalves (ICMC/USP)

ÍNDICE | *TABLE OF CONTENTS*

Análise Histórica do WDDS/WDES

Rodrigo Santos[1], Ivaldir Farias Junior[2], Thaiana Lima[1], Luisa Hernández[3],

[1] PESC/COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

[2] CIn – Universidade Federal de Pernambuco (UFPE)

[3]DCC – Universidade Federal de Lavras (UFLA)

{rps, thaiana}@cos.ufrj.br, {ivaldirjr, lufe.hernandez}@gmail.com

Abstract. *The Workshop on Distributed Software Development (WDDS), now Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (WDES), has been established as a forum for researches related to technical, business and social aspects of Software Engineering. This paper presents an analysis of the WDDS and WDES proceedings from 2007 to 2014. We analyzed a total of 85 papers and an h-index analysis from Google Scholar was performed. The papers were also classified into different perspectives to support discussions on the future of the workshop.*

Resumo. *O Workshop de Desenvolvimento Distribuído de Software (WDDS), agora Workshop em Desenvolvimento Distribuído de Software, Ecossistemas de Software e Sistemas-de-Sistemas (WDES), tem se firmado como um fórum para pesquisas relacionadas a aspectos técnicos, econômicos e sociais da Engenharia de Software. Este trabalho apresenta uma análise dos anais do WDDS e WDES de 2007 a 2014. Analisamos um total de 85 artigos e foi realizada uma análise do h-index a partir do Google Scholar. Os trabalhos foram ainda classificados para apoiar discussões sobre o futuro do workshop.*

1. Introdução

Conforme apresentado na chamada de trabalhos do WDES (2015), o Desenvolvimento Distribuído de Software (DDS) refere-se a uma abordagem de desenvolvimento que visa utilizar recursos distribuídos, com o objetivo de diminuir custos e atender a novos mercados. A distribuição da equipe de desenvolvimento e a dificuldade de coordenação e comunicação por vezes se traduzem em processos lentos, culminando em problemas inerentes à Engenharia de Software. Em paralelo, uma classe de sistemas de software tem surgido: os Sistemas-de-Sistemas (SoS). SoS se referem a sistemas maiores e mais complexos. Independência operacional e gerencial, desenvolvimento evolucionário, distribuição geográfica dos sistemas constituintes e comportamento emergente evidenciam teorias e práticas do DDS aplicáveis no desenvolvimento de SoS. Em outra perspectiva, os Ecossistemas de Software (ECOS) consistem em um conjunto de atores que interagem em um mercado de software e serviços, cujas relações são apoiadas por uma plataforma tecnológica. Com o desafio de tratar questões econômicas e sociais integradas às questões técnicas de desenvolvimento, ECOS vêm se tornando um importante tópico na Engenharia de Software, com impactos em DDS e em SoS.

Instanciando o trabalho de Novais *et al.* (2014), que buscou dar visibilidade às publicações dos eventos VEM, WBVS e WMSWM, a nossa pesquisa busca unir competências e abranger as pesquisas das áreas do Workshop em Desenvolvimento Distribuído de Software, Ecossistemas de Software e Sistemas-de-Sistemas (WDES), até então Workshop de Desenvolvimento Distribuído de Software (WDDS). Em sua nona edição, o evento traz, como principal tema em 2015, "Impactos de Ecossistemas de Software sobre a Qualidade no Desenvolvimento Distribuído de Software e em Sistemas-de-Sistemas". Este evento constitui um fórum para discussão de resultados e experiências de pesquisadores e praticantes das áreas de DDS, SoS e ECOS, visando a geração de conhecimento que possa viabilizar projetos de software de sucesso. O workshop almeja ampliar as possibilidades de colaboração em âmbito nacional e internacional, assim como consolidar as pesquisas em DDS, ECOS e SoS como uma subárea estratégica da Engenharia de Software no Brasil, como acontece no exterior.

Em 2015, paralelamente a um esforço para o estabelecimento do workshop em sua nova visão, buscamos o seu registro no sistema Qualis CAPES[1], por meio da análise do *h-index* extraído do *Google Scholar*, pois eventos em geral são enquadrados na categoria "conferência" [CAPES, 2013]. O objetivo deste artigo é, então, apresentar uma análise histórica do WDDS e WDES no período 2007-2014. Analisamos um total de 85 artigos, classificados em diferentes perspectivas a partir da análise dos seus metadados obtidos nos anais. Isso permite uma visão geral das pesquisas desenvolvidas nos últimos oito anos a fim de apoiar discussões sobre o futuro do workshop. Outro resultado foi a análise do *h-index* a partir da tabulação das citações feitas aos artigos do workshop. O artigo está organizado nas seguintes seções: a Seção 2 resume o histórico de organização do WDDS/WDES; a Seção 3 expõe os resultados da análise do *h-index* do workshop; a Seção 4 apresenta o planejamento e resultados da análise histórica do workshop; por fim, a Seção 5 conclui o artigo com algumas considerações finais.

2. Histórico da Organização do WDDS/WDES

O WDDS teve a sua primeira edição em 2007, como parte integrante do Simpósio Brasileiro de Engenharia de Software (SBES), e contou com a participação de pessoas da academia e da indústria. Em 2008, a novidade foi um painel academia-indústria, visando integrar essas perspectivas e discutir os desafios de DDS. A edição de 2009 apresentou o melhor desempenho do WDDS até o momento, como mostra a Tabela 1. Outra importante iniciativa foi convidar os autores dos melhores artigos das edições 2007 a 2009 para publicarem versões estendidas em edições especiais do periódico INFOCOMP[2]. Em 2010, o workshop passou a ser realizado junto com o Congresso Brasileiro de Software: Teoria e Prática (CBSoft). Em 2012, com a vinda do IEEE *International Conference on Global Software Engineering* (ICGSE)[3] ao Brasil, a comunidade de DDS enxergou uma oportunidade para estreitar relações com a comunidade internacional e o WDDS foi realizado em conjunto com ICGSE.

Em 2012, constatou-se a participação de novos pesquisadores e estudantes de mestrado e doutorado. O principal resultado foi o estreitamento das relações entre os pesquisadores, em especial, com aqueles de ECOS, que identificaram a sobreposição de temas de interesse. Na edição de 2013, foi realizada uma palestra intitulada "Desafios de DDS e a abordagem do grupo Volvo", a fim de manter aproximação com a indústria. Além disso, como fruto dos encaminhamentos da edição de 2012, o WDDS estabeleceu uma parceria com a comunidade de ECOS e três artigos deste tema foram selecionados

[1] Sistema WebQualis. Disponível em: <http://qualis.capes.gov.br>
[2] INFOCOMP – *Journal Computer Science*. Disponível em: <www.dcc.ufla.br/infocomp/>
[3] IEEE ICGSE. Disponível em: <http://www.icgse.org/>

para apresentação em sessões técnicas. No encerramento, a Profa. Cláudia Werner foi convidada para realizar uma apresentação, apontando possíveis interseções entre DDS x ECOS, ECOS x SoS e DDS x SoS, e sugeriu a introdução de um tema novo ao evento, denominado SoS, para que as comunidades pudessem colaborar em suas interseções. Como consequência, na edição de 2014, um ajuste no nome do workshop foi realizado, passando a se chamar WDES.

Tabela 1. Histórico da organização do workshop

EDIÇÃO	REALIZAÇÃO	SUBMISSÕES	ACEITAÇÕES	PARTICIPANTES
2007	SBES	27	12	40
2008	SBES	24	12	30
2009	SBES	28	10	35
2010	CBSoft	18	13	20
2011	CBSoft	12	7	20
2012	IEEE ICGSE	8	6	11
2013	CBSoft	21	12 (+ 5 pôsteres)	28
2014	CBSoft	6	4 (+ 4 artigos convidados)	25

3. Análise do *h-index* do WDDS/WDES

Com o intuito de buscar o registro do workshop no sistema Qualis CAPES, foi realizado o registro de 25 artigos dos dois últimos anos na Biblioteca Digital Brasileira de Computação[4] (2013[5] e 2014[6]), pois os artigos das edições anteriores a 2013 já estavam cadastrados. Isso viabilizou o processo de análise da quantidade de citações que cada artigo possuía no *Google Scholar*. Foi aplicado também o método *snowballing*[7] sobre os artigos do evento para complementar o processo. Após este trabalho, foi possível identificar o *h-index* do WDES, levando em consideração o histórico do WDDS.

Até junho de 2015, foram levantados os seguintes dados: 1 artigo com 22 citações; 1 artigo com 18 citações; 1 artigo com 13 citações; *3 artigos com 5 citações*; 5 artigos com 4 citações; 4 artigos com 3 citações; 11 artigos com 2 citações; 13 artigos com 1 citações; 46 artigos com nenhuma citação. Isto implica em um *h-index* igual a *5*. A partir do referencial estabelecido do Documento de Área 2013 – Ciência da Computação [CAPES, 2013], o workshop seria classificado na área (*b*) *Sistemas de Computação: Engenharia de Software* e teria um Qualis *B4*.

4. Análise Histórica do WDDS/WDES

4.1. Planejamento

Assim como foi realizado por Novais *et al.* (2014), a análise histórica do workshop foi planejada à luz das boas práticas do protocolo apresentado em [Kitchenham & Charters 2007]. Os itens do protocolo considerados importantes para a análise histórica são:

- *Questão de Pesquisa*:
    - Como foi a trajetória das publicações do WDDS/WDES de 2007 a 2014?

[4] Biblioteca Digital Brasileira de Computação. Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp>
[5] Anais do WDDS 2007-2013
Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/PesquisaEvento?evento=wdds>
[6] Anais do WDES 2014. Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Evento?id=787>
[7] O método *snowballing* consiste em analisar recursivamente as referências das referências de um dado artigo, até que um critério de parada seja atingido; por exemplo, nenhum resultado relevante para o objetivo do estudo realizado seja obtido.

- *Critérios de Inclusão e Exclusão*:
    - Foram incluídos os artigos do WDDS (2007-2013) e do WDES (2014).
- *Extração de Dados e Classificação*:
    - Foi utilizado um formulário de extração dos seguintes dados para cada artigo: autores, título, resumo, ano, instituição, idioma e citações (*Google Scholar* e *snowballing*). Dois pesquisadores categorizaram cada um dos artigos e dois pesquisadores revisaram os resultados.

4.2. Resultados

Na Tabela 1, pode-se verificar a distribuição dos artigos publicados por ano (2007-2014) e por evento (WDDS-WDES). Vale a pena destacar a edição de 2007 com 12 artigos aceitos (27 submetidos), 2010 com 13 artigos aceitos (18 submetidos) e 2013, quando o evento teve o maior número de artigos aceitos em uma única edição (12 completos e 5 pôsteres, de um total de 21 submetidos). Além disso, a edição de 2009 teve, até então, o maior número de submissões (28). Em contrapartida, a edição de 2011 teve somente 7 artigos aceitos (12 submetidos), 2012 com 6 artigos aceitos (8 submetidos) e 2014 com 4 artigos aceitos (6 submetidos) e 4 artigos convidados. A média de artigos publicados é de 10,6 por ano no workshop.

A queda nas submissões de artigos para o workshop nos anos de 2011, 2012 e 2014 pode ser explicada por algumas variáveis que tem impactado outros workshops, tais como o estímulo que as universidades têm dado aos seus alunos para publicarem em eventos que constem no Qualis CAPES, bem como o foco atual da área de Ciência da Computação em periódicos. Entretanto, é notório que o evento gera uma contribuição significativa para a comunidade científica.

Na Figura 1, pode ser observada a distribuição de artigos publicados por idioma, no caso, Português e Inglês. Em 2012, dado que o workshop foi realizado como evento satélite do IEEE ICGSE, o idioma oficial era Inglês, com um total de 6 artigos publicados. Em 2014, já renomeado para WDES, o workshop teve 5 artigos em Inglês publicados. Pela característica de workshop brasileiro, a preferência pelo Português foi observada, embora a comunidade tenha se esforçado para escrever os artigos em Inglês com o intuito de ter maior visibilidade das pesquisas realizadas.

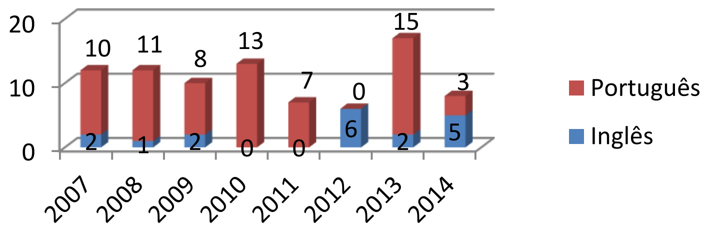


Figura 1. Distribuição da quantidade de artigos por idioma

A Figura 2 mostra a distribuição das publicações em relação à região dos autores envolvidos. Dos 14 estados que possuem grupos de pesquisa nos temas do workshop, Rio Grande do Sul (23), Pernambuco (23), Paraná (15), São Paulo (12) e Rio de Janeiro (10) apresentam maior participação em número artigos. Dois artigos dos EUA tiveram colaborações de universidades brasileiras: um artigo da *University of Maryland* com a Universidade de São Paulo [Malheiros *et al.* 2010] e um artigo da *Clemson University* com o Instituto Federal de Educação, Ciência e Tecnologia da Bahia [McGregor &

Amorim 2014]. Houve dois artigos entre a Universidade de São Paulo e universidades da França (*Institut de Recherche en Informatique et Systèmes Aléatoires*) e da Espanha (*Universidad Rey Juan Carlos*) [Nakagawa *et al.* 2014] [Santos *et al.* 2014a].

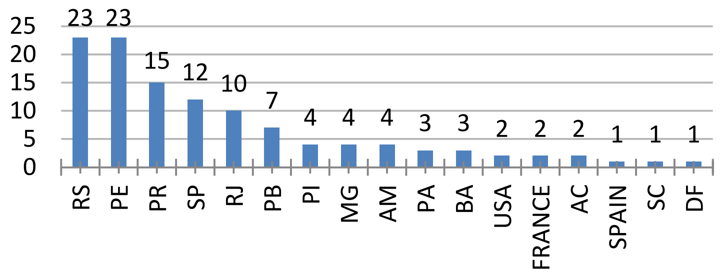


Figura 2. Distribuição da quantidade de artigos por região de origem dos autores

A Figura 3 mostra a distribuição das publicações do workshop em relação às instituições brasileiras com, pelo menos, dois artigos. No total, foram identificadas 39 instituições, das quais 23 possuem uma única publicação. A Universidade Federal de Pernambuco apresenta a maior quantidade de publicações (23), seguida pela Pontifícia Universidade Católica do Rio Grande do Sul (22), Universidade Federal do Rio de Janeiro (10), Universidade Estadual de Maringá (10) e Universidade de São Paulo (8).

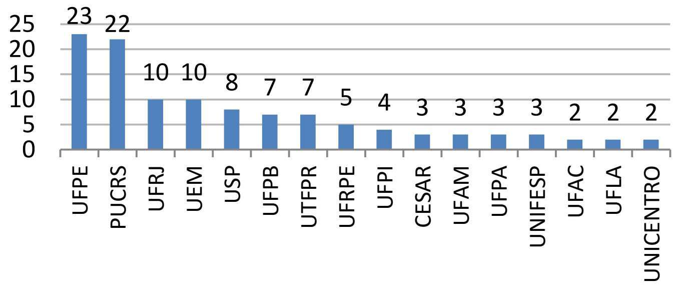


Figura 3. Distribuição da quantidade de artigos por instituição de origem dos autores

Com relação à análise das citações de cada artigo publicado no workshop, a Figura 4 mostra os resultados obtidos a partir do *Google Scholar*. A coleta de dados foi realizada em julho de 2015 (atualizada em julho de 2015). Observa-se que as pesquisas apresentadas ao longo das edições do workshop têm servido de base para a comunidade de pesquisadores nos temas. Outro aspecto importante foi o impacto do WDDS 2012, realizado com o IEEE ICGSE, cujos anais (em Inglês) foram publicados na *IEEE Xplorer Digital Library*[8]. Acreditamos que isso contribuiu para um número maior de citações das publicações de 2012, ficando atrás somente da primeira edição (2007).

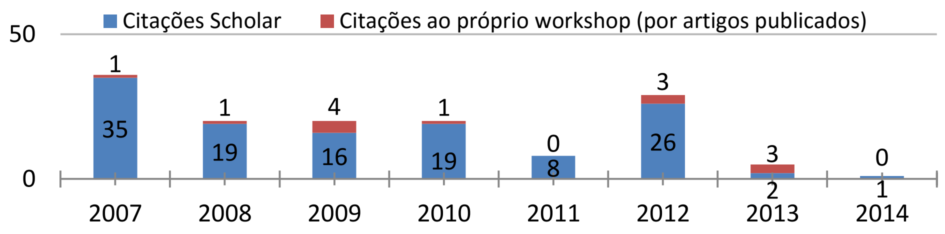


Figura 4. Distribuição do número de citações dos artigos

Ainda na Figura 4, percebemos que é baixo o número de citações feitas a artigos publicados no próprio workshop, destacando-se as edições de 2009 (4 citações), 2012 e

[8] Disponível em: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6336440>

2013 (3 citações cada). Essa informação deve servir de alerta para a comunidade, uma vez que o workshop tem um papel importante na discussão e evolução do estado da arte dos temas relacionados. Isso também impacta o *h-index* do workshop e, consequentemente, a sua avaliação Qualis CAPES. Por outro lado, várias colaborações entre instituições (universidades, empresas e centros de pesquisa) foram observadas nas publicações do workshop, como mostra o grafo da Figura 5. Foram contabilizados todos os artigos aceitos entre 2007 e 2014 e extraídas as instituições envolvidas.

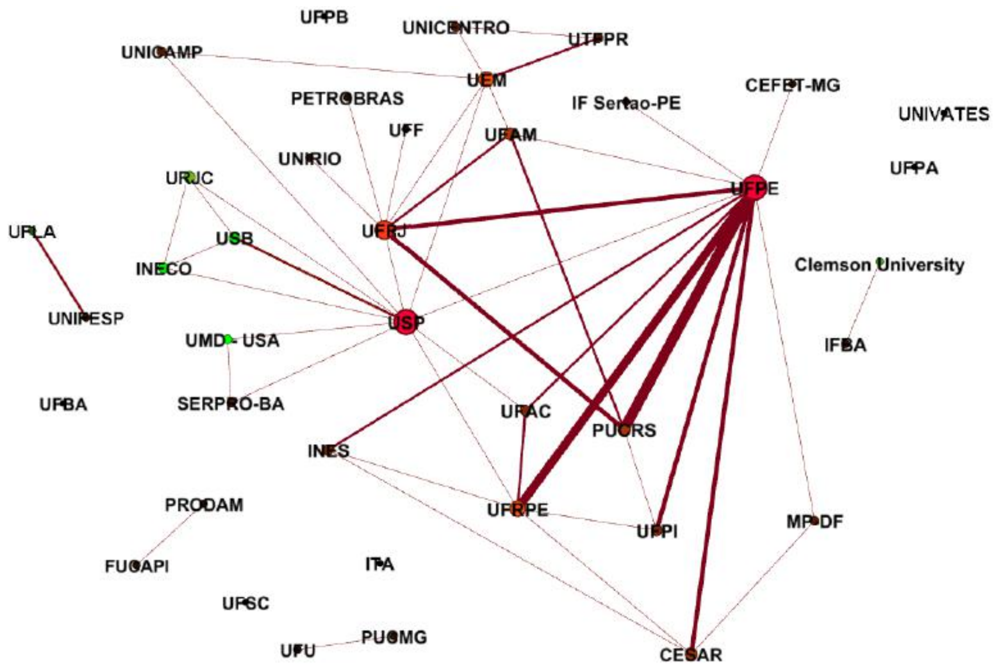


Figura 5. Grafo de colaboração entre instituições

Cada nó do grafo representa uma instituição. As arestas exibem relacionamentos entre instituições que colaboraram em publicações. O tamanho dos nós é proporcional à quantidade de artigos publicados em colaboração, assim como a espessura da aresta. Os nós marcados na cor verde são instituições internacionais. Entre as instituições que mais colaboram, destacam-se: UFPE, USP, UFRJ, UEM, UFRPE, PUCRS, UFAM, CESAR, UFPI, UFAC e INES. Torna-se assim importante que os pesquisadores envolvidos nos temas participem do workshop ativamente a fim de amadurecer trabalhos em andamento e gerar colaborações em projetos de pesquisa e parcerias em futuras publicações.

Visando identificar algumas das expressões mais utilizadas nas publicações do workshop em todas as suas edições, uma nuvem de palavras foi construída com base em parte dos textos (resumos/*abstracts* e palavras-chave). Inicialmente, esses textos foram extraídos manualmente dos arquivos em formato PDF e, então, convertidos para um único arquivo no formato TXT. Por fim, a ferramenta *Tagxedo*[9] foi utilizada para gerar a nuvem de palavras, cujo resultado é apresentado na Figura 6. A ferramenta fornece uma lista contendo cada palavra do texto previamente carregado. No entanto, a seleção das palavras que se deseja omitir (e.g., *stop words*) é totalmente manual, o que dificultou a filtragem de conectores como "e" ou palavras/expressões como "portanto", "no entanto" etc.; neste caso, fez-se uma seleção manual para omiti-las. A nuvem de palavras produzida pode ser utilizada para promover discussão na comunidade de

[9] Tagxedo. Disponível em: <http://www.tagxedo.com/app.html>

pesquisa nos temas relacionados, a partir dos assuntos que mais se destacaram nos últimos anos. Por exemplo, *desenvolvimento*, *ambiente*, *modelo*, *desafios*, *equipes*, *comunicação*, *ferramentas*, *artefatos*, *projetos*, *processos*, *práticas*, *testes*, entre outros.

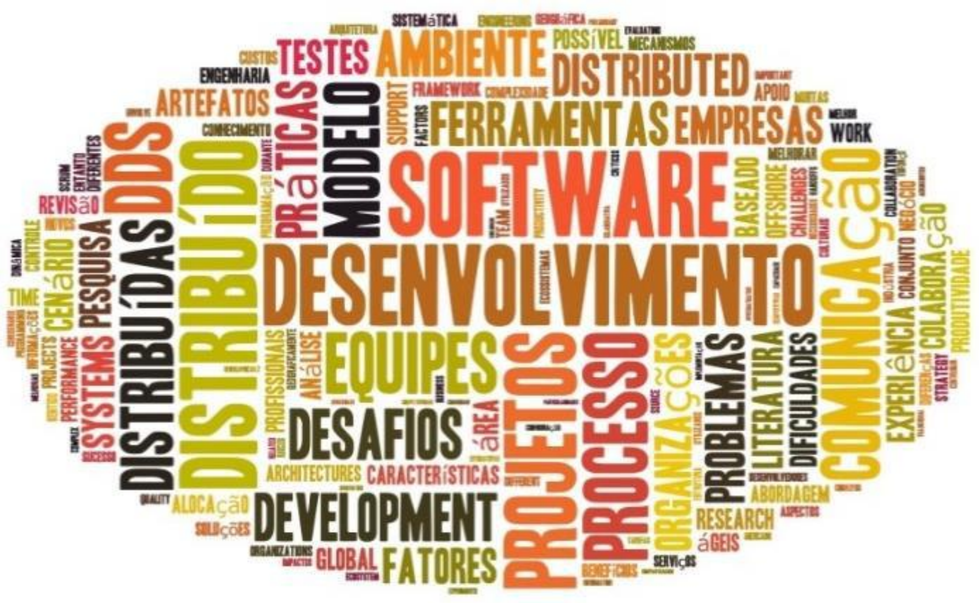


Figura 6. Nuvem de palavras produzida a partir dos resumos/abstracts e palavras-chave

Por fim, analisamos as publicações do workshop entre 2007 a 2014 e, em seguida, classificamos os artigos publicados considerando as 10 áreas estabelecidas pelo SWEBOK[10]. A Figura 7 mostra a linha do tempo das publicações do workshop sob essas áreas. Em 2007, a área mais pesquisada conforme as publicações do workshop foi *Processo de Engenharia de Software*. Entretanto, o enfoque nesta área foi reduzido nos anos seguintes, o que pode ter sido ocasionado pelo aumento do interesse na área *Gerência de Engenharia de Software*. Desde 2008, esta área têm se mantido como a mais pesquisada segundo as publicações do workshop. Por outro lado, a área *Qualidade de Software* não foi objeto de pesquisa dentro das publicações do evento. Vale salientar que, até 2013, a área mais pesquisada foi DDS, com o maior número de publicações, mesmo com a introdução de ECOS em 2012. Entretanto, em 2014, o evento teve uma distribuição mais equilibrada entre as áreas DDS, ECOS e SoS, com pelo menos uma publicação cada.

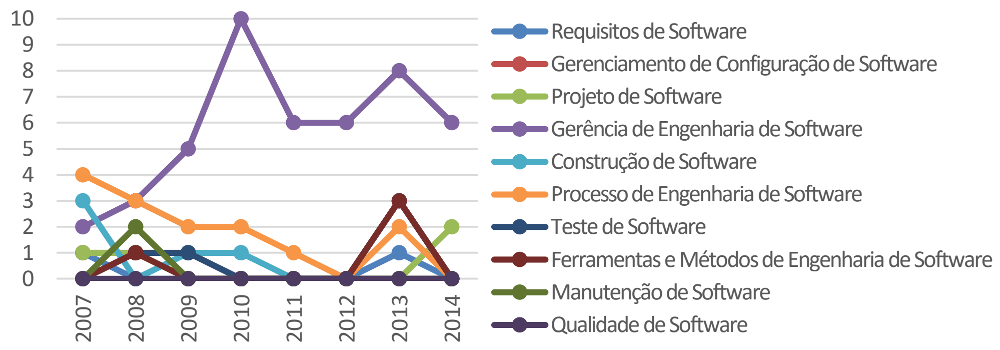


Figura 7. Quantidade de artigos publicados no workshop por área do SWEBOK

[10] Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Disponível em: <http://www.computer.org/web/swebok>

5. Considerações Finais

O WDDS, agora WDES, se constitui como um fórum para o debate de experiências de pesquisadores e de praticantes de novos temas de Engenharia de Software, visando gerar e consolidar conhecimentos a serem utilizados para adoção, avaliação e execução de projetos de DDS, ECOS e SoS no Brasil. Deve-se mencionar que o estudo de soluções tecnológicas, organizacionais e sociais advindo do workshop está alinhado com as políticas públicas do Ministério da Ciência, Tecnologia e Inovação (MCTI), que visam à melhoria da qualidade dos processos, dos produtos e dos serviços de software brasileiros, de modo a tornar as empresas mais capacitadas a competir no mercado globalizado. É importante destacar que a combinação de soluções produzidas nas pesquisas em DDS, ECOS e SoS pode e deve ser explorada, uma vez que os três tipos de sistemas compartilham algumas características [Santos *et al.* 2014bc].

Neste artigo, foi apresentado parte do esforço que tem sido despendido em 2015 para fortalecer o workshop, como: (1) a disponibilização dos anais de todas as edições na Biblioteca Digital Brasileira de Computação; (2) a análise do *h-index* do workshop a partir do Google Scholar; e (3) a análise histórica do workshop a partir da classificação de um total de 85 artigos em diferentes perspectivas. Após oito anos de workshop, conclui-se que são necessárias ações da comunidade de pesquisa dos temas envolvidos: (1) maior divulgação do workshop (submissão de artigos); (2) realização de painel com acadêmicos e profissionais da indústria (interação com a realidade); (3) realização de palestras nacionais e internacionais (novos conhecimentos); (4) desenvolvimento de grupos de trabalho específicos para identificar e discutir questões de pesquisa (colaboração); e (5) participação de pesquisadores nacionais e estrangeiros (presença da comunidade). Pretende-se realizar uma pesquisa de opinião com os pesquisadores dos temas a fim de identificar potenciais colaborações e desafios de pesquisa.

Referências

CAPES (2013) "Documento de Área 2013 – Ciência da Computação". Disponível em: <https://www.capes.gov.br/images/stories/download/avaliacaotrienal/Docs_de_area/Ci%C3%AAncia_da_Computa%C3%A7%C3%A3o_doc_area_e_comiss%C3%A3o_att08deoutubro.pdf>

Kitchenham, B., Charters, S. (2007) "Guidelines for Performing Systematic Literature Reviews in Software Engineering". TR-EBSE-2007-01, Keele University & Durham University.

Malheiros, V., Seaman, C., Maldonado, J. (2010) "An Approach for Collaborative and Distributed Software Process Improvement (SPI)". In: XXIII SBES, III WDDS, Fortaleza, pp. 21-30.

McGregor, J., Amorim, S. (2014) "Ecosystem Business Models and Architectures". In: V CBSOFT, VIII WDES, Maceió, pp. 33-40.

Novais, R., Mendes T., Teles, F. (2014) "Uma Análise da História do VEM, WBVS e WMSWM". In: V CBSOFT, II VEM, Maceió, pp. 118-125.

Nakagawa, E., Capilla, R., Díaz, F., Oquendo, F. (2014) "Towards the Dynamic Evolution of Context-based Systems-of-Systems". In: V CBSOFT, VIII WDES, Maceió, pp. 45-52.

Santos, D., Oliveira, B., Guessi, M., Oquendo, F., Delamaro, M., Nakagawa, E. (2014a) "Towards the Evaluation of System-of-Systems Software Architectures". In: V CBSOFT, VIII WDES, Maceió, pp. 53-57.

Santos, R., Valença, G., Viana, D., Estácio, B., Fontao, A., Marczak, S., Werner, C., Alves, C., Conte, T., Prikladnicki, R. (2014b) "Qualidade em Ecossistemas de Software: Desafios e Oportunidades de Pesquisa". In: V CBSOFT, VIII WDES, Maceió, pp. 41-44.

Santos, R., Gonçalves, M., Nakagawa, E., Werner, C. (2014c) "On the Relations between Systems-of-Systems and Software Ecosystems". In: V CBSOFT, VIII WDES, Maceió, pp. 58-62.

WDES (2015) "Workshop em Desenvolvimento Distribuído de Software, Ecossistemas de Software e Sistemas-de-Sistemas". Disponível em: <http://wdes2015.icmc.usp.br>

Um Estudo Exploratório Sobre Contribuições Casuais em Projetos de Software Livre: Caso do Projeto LibreOffice

Felipe V. Ramos[1], **Marco Aurélio Gerosa**[2], **Ana Paula Chaves**[1], **Igor Steinmacher**[1]

[1]Departamento de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR) – Campo Mourão-PR

[2]Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP) – São Paulo-SP

fveigaramos@gmail.com, gerosa@ime.usp.br, {anachaves, igorfs}@utfpr.edu.br

Abstract. *Many free software communities leverage contributions from volunteers and need to attract, motivate and engage new developers. A phenomenon that has attracted attention are the casual contributions made by developers who are not interested in becoming project members. The goal of this paper is to conduct an exploratory study of the interaction of newcomers to LibreOffice to understand the phenomenon of casual contributions. We analyzed historical data from mailing lists, issue tracker and code review management tool. The results showed that 27.1% of the 2013 newcomers were classified as casual contributors and the effort by the community to support them is relatively low.*

Resumo. *Diversas comunidades de software livre contam com contribuições de voluntários e precisam atrair, motivar e engajar novos desenvolvedores. Um fenômeno que tem atraído atenção são as contribuições casuais, realizadas por desenvolvedores que não tem interesse em se tornar membros dos projetos. O objetivo deste artigo é conduzir um estudo exploratório da interação dos novatos do LibreOffice para compreender o fenômeno das contribuições casuais. Para isso, foram analisados dados históricos da lista de emails, gerenciador de tarefas e gerenciador de revisões de código. Os resultados mostram que 27,1% dos novatos de 2013 foram classificados como contribuintes casuais e o esforço despendido pela comunidade para auxiliá-los é relativamente baixo.*

1. Introdução

Uma das aplicações mais bem sucedidas da abordagem de desenvolvimento distribuído de software é o modelo utilizado por comunidades de software livre [Carmel and Tjia 2005, German 2003]. Muitos desses projetos são comunidades de contribuição aberta, dependendo de contribuições de voluntários para sua manutenção, evolução e sustentabilidade. Sendo assim, é essencial motivar, engajar e reter novos desenvolvedores.

Existem diversos estudos na literatura focando em diferentes aspectos da entrada de novatos, incluindo como se tornar um membro, motivação para entrada e retenção [Steinmacher et al. 2015b]. Entretanto, as pesquisas anteriores se preocupam em entender a dinâmica que leva os novatos a tornarem-se membros, incluindo desenvolvedores do núcleo e desenvolvedores de longo prazo [Fagerholm et al. 2014, Steinmacher et al. 2015a]. Um tópico negligenciado pela literatura diz respeito aos desenvolvedores que não tem interesse em se tornar membros dos projetos, mas contribuir apenas uma vez ou resolver um problema pontual. Esse tipo de comportamento é chamado de "*drive-by commit*" [Pham et al. 2013], e aqui será chamado de contribuições casuais.

O fenômeno das "contribuições casuais" vem ganhando atenção recentemente. No GitHub[1], por exemplo, essas contribuições tem se tornado mais comuns [Pham et al. 2013]. De acordo com Gousios et al. [Gousios et al. 2014], este tipo de contribuição foi responsável por 7% dos *pull requests* no GitHub em 2012. Essas contribuições casuais possibilitam maior colaboração e mais diversidade no projeto [Pham et al. 2013] e não dependem de engajamento específico com o projeto e, muitas vezes, são feitas por colaboradores que não voltarão.

De acordo com a literatura corrente, mais pesquisas relacionadas ao fenômeno das contribuições casuais precisam ser realizadas para um melhor entendimento dos processos, dos benefícios [Pham et al. 2013] e das implicações [Gousios et al. 2014, Vasilescu et al. 2015] destas contribuições. Visto isso, o objetivo do presente artigo é conduzir um estudo exploratório preliminar da interação dos contribuintes novatos do LibreOffice para compreender o fenômeno das contribuições casuais nesse projeto. Para alcançar tal objetivo, foram definidas as seguintes questões de pesquisa:

Q1. É possível encontrar indícios da existência de contribuintes casuais?
Q2. Qual o esforço que tais contribuições demandam da comunidade?

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados; a Seção 3 descreve a forma como os dados foram coletados e analisados; as Seções 4 e 5 discutem os resultados e as limitações desta pesquisa, respectivamente; as conclusões e possíveis desdobramentos são apresentados na Seção 6.

2. Trabalhos Relacionados

Vários trabalhos na literatura analisam aspectos específicos da entrada de novatos em projetos de software livre [Park and Jensen 2009, von Krogh et al. 2003, Steinmacher et al. 2015b, Zhou and Mockus 2015, Ducheneaut 2005, Bird 2011, Jensen et al. 2011, Stol et al. 2010]. Park e Jensen [Park and Jensen 2009] estudaram as necessidades dos novatos por informação. Os autores mostram que ferramentas de visualização de informações apoiam os primeiros passos de novatos na aprendizagem sobre um projeto de software livre, ajudando-os a encontrar informações mais rapidamente. Os autores perceberam que os novatos apoiados por ferramentas de visualização concluem suas atividades mais rapidamente e compreendem melhor o código do projeto. Por sua vez, Von Krogh et al. [von Krogh et al. 2003] conduziram um estudo sobre o projeto FreeNet, por meio de entrevista com desenvolvedores, análise de histórico de emails, repositório de código fonte e documentos do projeto. Os autores propuseram um roteiro de entrada para desenvolvedores. Uma de suas contribuições indica que os novatos frequentemente ficam observando o projeto antes de iniciar sua participação, para, em seguida, interagir. Apesar de estudar o processo de entrada em projetos de software livre, não há preocupação com a análise das razões de desistência, verificando apenas o comportamento daqueles que se tornaram membros do projeto.

Alguns trabalhos [Zhou and Mockus 2015, Ducheneaut 2005, Bird 2011] também analisam como a socialização influencia na retenção dos novatos em projetos de software

[1]http://www.github.com

livre. Esses trabalhos analisam as redes sociais de comunicação (listas de email) para verificar com quem os novatos se relacionam, como a rede de contatos evolui de acordo com o tempo e como ocorre o relacionamento com os outros membros do projeto. Esses estudos, entretanto, mais uma vez analisam apenas o comportamento daqueles que permanecem no projeto e se tornam desenvolvedores do núcleo, sem analisar os problemas dos novatos que desistem ou que permanecem na periferia.

Jensen et al. [Jensen et al. 2011] fazem uma análise de quatro listas de emails de projetos de software livre, visando verificar se os emails de novatos são respondidos rapidamente, se o sexo e a nacionalidade dos novatos interferem no tipo de resposta recebida nos emails e na continuidade dos novatos, e, por fim, se o tratamento a novatos é diferente na lista de usuários e na lista de desenvolvedores. Por sua vez, Stol et al. [Stol et al. 2010] conduziram um estudo relacionado à identificação de padrões arquiteturais em projetos de software livre. O estudo de caso foi conduzido por novatos, que, ao final da análise foram entrevistados. Esses novatos relataram um conjunto de problemas e obstáculos enfrentados por eles durante os primeiros contatos com o projeto.

Nakakoji et al. [Nakakoji et al. 2002] estudam quatro projetos de software livre para analisar a evolução da comunidade desses projetos. O estudo apresenta oito papéis encontrados para os membros de um projeto e os apresenta na forma de uma cebola, o chamado *onion patch*. A hipótese desse modelo indica que novatos geralmente iniciam pelas camadas mais externas do modelo e vão em direção ao centro de acordo com seus objetivos. Esses artigos tratam da entrada e evolução da participação de membros em comunidades de software livre, mas nenhum dos estudos foca em novatos que desejam realizar contribuições casuais e não se tornar membros dos projetos.

Com relação às contribuições casuais, diversos autores [Pham et al. 2013, Gousios et al. 2014, Vasilescu et al. 2015] citam em suas pesquisas a existência e crescimento desse fenômeno, mas nenhum dos trabalhos endereça diretamente essa questão. Os autores destacam que é importante observar o impacto destas contribuições para os projetos e os benefícios que trazem para a comunidade, apontando o tópico como um desafio em aberto.

Dada a recente identificação do fenômeno das contribuições casuais, este trabalho tem por objetivo agregar conhecimento ao estado da arte estudando quantitativamente possíveis casos de contribuições casuais no projeto LibreOffice.

3. Método de Pesquisa

Para analisar as contribuições casuais do projeto LibreOffice, foram coletados dados históricos e procedeu-se a análise dos dados de acordo com as questões de pesquisa. A seguir são apresentados os procedimentos de coleta e análise adotados.

Foram coletados dados do projeto LibreOffice, a partir da lista de email[2], software gerenciador de tarefas (Bugzilla[3]), sistema de revisão de código (Gerrit[4]) e os registros (logs) do repositório de código-fonte[5]. Para realizar a coleta de dados foram utilizadas ferramentas da suíte MetricsGrimoire[6], a saber:

[2]http://lists.freedesktop.org/archives/libreoffice/
[3]https://bugs.documentfoundation.org/
[4]https://gerrit.libreoffice.org
[5]git://anongit.freedesktop.org/libreoffice/core
[6]https://metricsgrimoire.github.io/

Table 1. Distribuição dos novatos por número de *patches*

patches enviados	# pessoas	% das pessoas	# de *patches* aceitos	% do total de *patches*
Até 2	19	27,14%	24	2,36%
Entre 3 e 6	17	24,29%	54	5,32%
Entre 7 e 23	17	24,29%	153	15,07%
Mais de 23	17	24,29%	784	77,24%

- Bicho: utilizada para recuperar e organizar as informações de sistemas gerenciadores de tarefas, neste caso, foi utilizado para o bugzilla e gerrit.
- MailingListStats: recupera os dados de listas de email. Para esta pesquisa, foi utilizada para baixar e armazenar os dados históricos das listas de email que estavam arquivados pelo projeto no formato mbox.

Foram coletados dados das fontes citadas, do período compreendido entre 1 de janeiro de 2012 e 31 de dezembro de 2014. Foram considerados novatos aqueles desenvolvedores que submeteram código pela primeira vez no ano de 2013. Para identificar tais novatos, foi construída uma ferramenta que percorre a lista de desenvolvedores e contribuintes do projeto[7] e analisa os emails referentes à declaração de licença (do inglês, *license statement*), buscando por aqueles datados de 2013. A declaração de licença consiste em mensagem enviada pelo desenvolvedor à lista de emails do projeto declarando que todas as suas contribuições estarão sob licença de software livre. Todos os desenvolvedores que enviaram a declaração de licença no ano de 2013 foram considerados novatos e tiveram suas interações com o projeto analisadas.

Para cada um dos novatos foram coletados os seguintes dados, que foram analisados para responder às questões de pesquisa:

- alterações de códigos (*patches*) enviados pelo novato para revisão a partir da ferramenta Gerrit;
- emails enviados pelo novato para a lista de desenvolvedores e respostas recebidas da comunidade;
- tarefas enviadas e comentários feitos pelo novato, a partir da ferramenta Bugzilla;
- *patches* enviados para revisão e comentários realizados nesses *patches*, a partir da ferramenta Gerrit;

Os dados dos emails, tarefas e revisão de código identificados foram inspecionados manualmente para auxiliar no entendimento do fenômeno no contexto do projeto.

4. Resultados

A coleta de dados retornou um total de 121 contribuintes considerados novatos. Foram desconsiderados 51 novatos que nunca enviaram código, resultando em 70 novatos. A Tabela 1 sumariza a distribuição dos novatos por número de *patches* enviados e também a quantidade total de patches aceitos por cada agrupamento.

4.1. Questão 1

Para responder à questão *"É possível encontrar indícios da existência de contribuintes casuais?"*, primeiramente foram comparadas a quantidade de solicitações de mudanças submetidas e a quantidade atribuída ao novato. Como essas quantidades eram idênticas,

[7]https://wiki.documentfoundation.org/Development/Developers/#Individuals

apenas os dados de submissões foram utilizados na análise. Para este estudo, foram considerados contribuintes casuais aqueles que enviaram até dois *patches* em 2013, sem enviar mais contribuições até o final de 2014.

Dos 70 novatos considerados, 11 submeteram um único *patch* (15,7% do total). Se considerarmos contribuintes casuais indivíduos que submeteram até dois *patches*, tem-se o total de 19 pessoas, ou 27,1% do total. É importante destacar que todas as pessoas que enviaram apenas um *patch* tiveram suas contribuições aceitas. Com relação às oito pessoas que enviaram duas contribuições, apenas três tiveram um *patch* aceito e um *patch* negado. Porém, durante a inspeção manual descobriu-se que, em um dos casos, seu *patch* negado estava duplicado em outra solicitação de mudança, sob nome de outro desenvolvedor. Portanto, apenas dois novatos tiveram apenas uma submissão aceita.

Assim, como resposta à Questão 1, foram encontrados indícios da existência de contribuintes casuais já que, para o caso do projeto LibreOffice, com novos contribuintes identificados em 2013, 27,1% do total de novatos enviaram 1 ou 2 propostas de *patch* e não retornaram ao projeto.

4.2. Questão 2

Para responder à questão *"Qual o esforço que tais contribuições demandam da comunidade?"*, foram analisadas a quantidade de interações que o novato realizou com membros da comunidade por meio da lista de emails, gerenciador de tarefas e ferramenta de revisão de códigos, além daquelas que fazem parte do processo de contribuição (como declaração de *statement*, revisão de código enviado e confirmação de aceite do *patch*).

Na Tabela 2 são apresentadas a quantidade de interações entre o novato e a comunidade. Para isso, foram analisadas a lista de emails e a ferramenta de revisão de código (gerrit). Não foram representadas as interações por meio do gerenciador de tarefas, pois, analisando essa ferramenta, não foram encontradas tarefas ou defeitos enviados pelos novatos e apenas um novato (C17) possuía tarefa atribuída a ele.

Analisando a quantidade de emails enviados, é possível observar que apenas quatro novatos (C5, C8, C14 e C17) enviaram mais de um email para a lista de desenvolvedores, sendo que três deles enviaram apenas 2, tendo sido respondidos por membros da comunidade (como pode ser visto na coluna "Respostas aos emails"). Foi conduzida uma análise manual, confirmando que todos os novatos enviaram a declaração de licença. Ainda durante a análise manual, identificou-se que: o novato C5 se apresentou para a comunidade e propôs uma nova funcionalidade; C8 pediu ajuda para iniciar, explicitamente solicitando auxílio na escolha de uma tarefa fácil; e C14 solicitou ajuda para compilar o código em seu ambiente local. Nos três casos, os contribuintes receberam duas mensagens de resposta da comunidade. O novato C17 foi o único caso diferenciado. Ele/ela propôs um projeto do *Google Summer of Code* em março de 2013 e trabalhou nesse projeto, tendo enviado emails relacionados ao projeto até outubro de 2013.

As duas últimas colunas da tabela representam a quantidade de mensagens trocadas na ferramenta de revisão de código nos *patches* enviados pelos novatos. Pode-se perceber que cada *patch* recebeu, no mínimo, dois comentários que, de acordo com o processo de revisão do projeto, é o número mínimo de mensagens possíveis. As mensagens correspondem a uma aprovação do revisor e a uma mensagem automática do *commit* para o sistema de versão. Dos 27 *patches* enviados para revisão, 15 (55.56%) receberam o

Table 2. Distribuição dos novatos por número de *patches*

Novato	Emails enviados	Respostas aos emails	*Patches* (gerrit)	Comentários nos *patches*	
				Outros membros**	Novato
C1	1	0	1	4	0
C2	1	0	1	2	0
C3	1	0	1	4	2
C4	1	0	1	4	1
C5	2	2	1	2	0
C6	1	0	1	2	0
C7	1	0	1	6	0
C8	2	2	1	2	1
C9	1	0	1	2	0
C10	1	0	1	3	0
C11	1	0	1	2	0
C12*	1	0	2	4 (4/2)	2
C13*	1	0	2	4 (2/2)	0
C14*	2	2	2	11 (3/8)	7
C15*	1	0	2	4 (2/2)	0
C16*	1	0	2	4 (2/2)	1
C17*	11	7	2	6 (4/2)	1
C18*	1	0	2	5 (2/3)	1
C19*	1	0	2	8 (4/4)	0

*contribuintes com 2 *patches* enviados

**os números entre parênteses representam a quantidade de comentários por *patch* enviado

número mínimo de comentários, o que significa que as contribuições estavam em conformidade. Na média, cada submissão recebeu 2,93 comentários. Houve apenas dois casos em que uma submissão rendeu mais de 4 comentários por parte da comunidade.

Por meio desta análise preliminar, percebeu-se que o esforço que as contribuições casuais demandam da comunidade é relativamente pequeno. Foram poucos os novatos que solicitaram ajuda pela lista de email e, dentre os que solicitaram, as respostas foram simples, direcionando-os a recursos existentes no wiki do projeto. Com relação à revisão dos códigos submetidos, apenas um *patch* submetido, dentre os 27 analisados, teve uma quantidade maior de interações (C14, com 8 comentários de membros da comunidade).

5. Limitações

Como em qualquer pesquisa empírica, este trabalho tem algumas limitações. Primeiramente, foi escolhido apenas um projeto e um período de tempo para conduzir a pesquisa. As conclusões e discussões apresentadas são específicas para o projeto. Sendo assim, não foi possível analisar a influência de variáveis como tamanho do projeto, quantidade de participantes, maturidade do projeto e domínio da aplicação. Para um resultado com conclusões para uma população mais ampla, é necessário analisar uma amostra significativa de projetos e períodos de verificação. Como trabalho futuro, pretende-se realizar uma pesquisa com outros projetos, incluindo projetos hospedados pelo Github.

Em segundo lugar, foram analisados apenas os dados dos projetos disponíveis *online* para realizar uma análise quantitativa. Para complementar a análise quantitativa, os emails trocados pelos novatos e as interações pelos outras ferramentas foram analisados manualmente. Uma pesquisa qualitativa utilizando dados coletados por meio de entrevistas com os contribuintes casuais pode revelar mais detalhes sobre esse comportamento.

As medidas utilizadas neste artigo podem não ser a melhor maneira de mostrar os resultados, podendo ser interpretada de maneiras diferentes. Não foram encontrados trabalhos que ofereçam outros meios para medição ou que possibilitem comparação ou confirmação dos resultados obtidos. Além disso, foram considerados contribuintes casuais aqueles desenvolvedores que submeteram até 2 *patches* para o projeto, já que não existem valores de referência na literatura que definam contribuintes casuais de acordo com a quantidade de *patches* submetidos por período de tempo.

As janelas de tempo escolhidas podem ter afetado as observações. Alterar o tamanho dos períodos de tempo ou alterar o início e fim dos intervalos de avaliação podem gerar observações diferentes. Usuários podem ter dois *logins* no gerenciador de tarefas ou participar da lista de desenvolvedores com dois emails diferentes. Pode, ainda, haver casos em que o mesmo usuário está cadastrado no gerenciador de tarefas e na lista de desenvolvedores com emails diferentes.

6. Conclusões

Neste artigo foi apresentado um estudo preliminar sobre o fenômeno das contribuições casuais, no escopo do projeto LibreOffice. Para este estudo buscou-se verificar a frequência desse tipo de contribuinte, bem como o esforço que tais contribuições demandam da comunidade. Foram analisados quantitativamente dados históricos extraídos da lista de emails de desenvolvedores, gerenciador de tarefas e gerenciador de revisões de código.

Considerando como contribuintes casuais aqueles desenvolvedores que enviaram até duas submissões de mudanças de código (*patches*) para o projeto LibreOffice no ano de 2013. Foram identificados 19 contribuintes que se encaixam nessa classificação, o que representa 27,14% dos 70 novatos identificados em 2013. Com relação ao esforço que essas contribuições demandam da comunidade, percebeu-se que poucos desses 19 novatos trocaram mensagens pela lista de emails, solicitando ajuda dos membros e também que, na média, a quantidade de mensagens solicitando alterações dos códigos submetidos ao sistema de revisão de códigos foi baixa.

Entende-se que as comunidades de software livre precisam utilizar o fenômeno das contribuições casuais a seu favor, beneficiando-se das possibilidades oferecidas. Visto que este fenômeno tem se tornado comum em projetos de software livre, é importante estudar o custo/benefício deste tipo de contribuição a fim de propor mudanças nos processos de contribuição de novatos de forma a facilitar e fomentar as contribuições.

Como trabalhos futuros, pretende-se entender que benefícios as contribuições casuais trazem para as comunidades. Pretende-se ainda conduzir estudos qualitativos, utilizando dados provenientes de entrevistas com contribuintes casuais e com membros da comunidade para melhor explorar o fenômeno. Além disso, estão sendo conduzidas pesquisas com projetos hospedados no Github, para aumentar a abrangência da pesquisa.

Agradecimentos

Os autores agradecem à Fundação Araucária, UTFPR, CNPq (processos universal 461101/2014-9 e 477831/2013-3), NAPSoL-PRP-USP e FAPESP pelo apoio financeiro.

References

Bird, C. (2011). Sociotechnical coordination and collaboration in open source software. In *Proc. of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 568–573. IEEE Computer Society.

Carmel, E. and Tjia, P. (2005). *Offshoring information technology: Sourcing and outsourcing to a global workforce*. Cambridge University Press.

Ducheneaut, N. (2005). Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work*, 14(4):323–368.

Fagerholm, F., Johnson, P., Guinea, A. S., Borenstein, J., and Munch, J. (2014). Onboarding in open source projects. *IEEE Software*, 31(6):54–61.

German, D. M. (2003). The gnome project: a case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201–215.

Gousios, G., Pinzger, M., and Deursen, A. v. (2014). An exploratory study of the pull-based software development model. In *Proc. of the 36th International Conference on Software Engineering*, ICSE 2014, pages 345–355. ACM.

Jensen, C., King, S., and Kuechler, V. (2011). Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists. In *Proc. of the 44th Hawaii International Conference on System Sciences*, HICSS '10, pages 1–10. IEEE.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. (2002). Evolution patterns of open-source software systems and communities. In *Proc. of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 76–85. ACM.

Park, Y. and Jensen, C. (2009). Beyond pretty pictures: Examining the benefits of code visualization for open source newcomers. In *Proc. of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '09, pages 3–10. IEEE.

Pham, R., Singer, L., Liskin, O., Figueira Filho, F., and Schneider, K. (2013). Creating a shared understanding of testing culture on a social coding site. In *Proc. of the 2013 International Conference on Software Engineering*, ICSE '13, pages 112–121. IEEE.

Steinmacher, I., Conte, T., Gerosa, M. A., and Redmiles, D. F. (2015a). Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proc. of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 1–13, New York, NY, USA. ACM.

Steinmacher, I., Silva, M. A. G., Gerosa, M. A., and Redmiles, D. F. (2015b). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85.

Stol, K.-J., Avgeriou, P., and Babar, M. A. (2010). Identifying architectural patterns used in open source software: approaches and challenges. In *Proc. of the 14th International conference on Evaluation and Assessment in Software Engineering*, EASE'10, pages 91–100. BCS.

Vasilescu, B., Filkov, V., and Serebrenik, A. (2015). Perceptions of diversity on github: A user survey. In *Proc. of the 2015 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '15. IEEE.

von Krogh, G., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: A case study. *Research Policy*, 32(7):1217–1241.

Zhou, M. and Mockus, A. (2015). Who will stay in the floss community? Modelling participant's initial behaviour. *IEEE Trans. Softw. Eng.*, 41(1):82–99.

Software Architecture Challenges in Distributed Development Settings: An Experience Report

Tassio Vale[1], Taslim Arif[2], Laia Gasparin[3]

[1]Federal University of Recôncavo da Bahia
Rua Rui Barbosa, 710, Centro - Cruz das Almas, Bahia, Brazil

[2]Fraunhofer- IESE
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

[3]VOMATEC Innovations GmbH
Riegelgrube 7, 55543 Bad Kreuznach, Germany

tassio.vale@ufrb.edu.br, taslim.arif@iese.fraunhofer.de,
laia.gasparin@vomatec-innovations.de

Abstract. *The RESCUER project proposes a system developed in a highly distributed setting of nine partners spread across the EU and Brazil. Regarding the software architecture activities in the project, failing to identify the key architectural challenges or identifying them at a very late stage of the project causes a lot of cost and effort. In this paper, we present a set of key architectural challenges identified during architecture iterations, and propose solution ideas to deal with it. Our experiences might benefit other organizations engaged in initiatives in these kinds of systems, since they can save valuable time and effort by discovering problems at a very early stage in the project.*

1. Introduction

Nowadays, most people use mobile devices and share their status and information about what is happening around them in real time. This phenomenon can help in an emergency situation, allowing a crowd with mobile devices to send detailed information to the command and control center. This information has high relevance for the operational forces because it is the key to what is happening in real time and originates from the place of the emergency.

The RESCUER system aims to develop a smart and interoperable information system that provides support in an emergency situation using crowdsourcing information. There are several challenges that we have experienced in architecting this system, such as: a) the user experience of mobile applications has to be excellent for diverse classes of users; b) in an emergency environment, the network is often interrupted and data collection therefore becomes difficult; c) the system has to be context-aware to make the analysis intelligent and visualization useful for the command and control center. On top of all these challenges, several development teams distributed throughout the EU and Brazil are developing our system. Such a distributed development setting raised the need for improved knowledge propagation on the technical level.

In this paper, we report a set of architectural challenges identified during the course of this project. We have also identified a set of solution concepts that could potentially help to tackle these challenges. We extend the report of the challenges and solution ideas from Vale et al. (2015), focusing on challenges concerning crowdsourcing systems.

Reported architectural challenges are the key for successful software development projects. Money and effort are saved if appropriate measures are taken to tackle these challenges. Building a crowd-based emergency management system involves a lot of functional and quality requirements. Moreover, distributed development settings create additional challenges during development and integration. We hope that our findings will make the architects of similar systems aware of all those traps that we learned to avoid over time in the hard way.

This paper is structured as follows: in Section 2, we describe the characteristics of the RESCUER project and part of its system architecture. Section 3 describes the architectural challenges and categorizes them according to the architectural viewpoints. In Section 4, we describe the high-level solution concepts and how we mapped them to our challenges. In chapter 5, we conclude our findings.

2. RESCUER Project

The RESCUER project aims to build a smart and interoperable computer-based solution to support emergency and crisis management, focusing on incidents in industrial areas and at large-scale events in Europe and Brazil. Such an infrastructure intends to provide faster and more accurate management in emergency and crisis situations by achieving: improved time to collect information regarding an emergency situation; decreased time and effort to analyze emergency data; improved and reliable information provided to different stakeholders within the shortest possible time; and context-aware interaction with different stakeholders; minimized effort among various workforces.

The goals and requirements defined in conjunction with the project stakeholders and partners guided the construction of the RESCUER architecture. We adopted the Fraunhofer ACES approach [Thorsten et al. 2011] as the software architecture construction process. It comprises three main activities: collecting requirements that need to be addressed by the system architecture; architecting the system; and documenting it through standard models.

The architecture document describes a set of architecturally significant requirements: development-related requirements, integration requirements, availability, robustness, scalability, reliability, performance, usability, security, safety, operability, upgradeability, auditability, and variability. Aiming to realize such requirements and to provide a better understanding for the software development teams, the RESCUER architecture was described in a set of perspectives. Regarding space constraints, we discuss in this paper only one of them: the sub-systems perspective.

2.1. Sub-Systems Perspective

The sub-systems perspective represents RESCUER as a set of components in order to decrease development complexity by assigning the construction of specific components to different teams, assuming they have the desired skills to do so. The RESCUER features are spread across five sub-systems: *Mobile Solution*, *Communication*

Infrastructure, *Data Analysis*, *Emergency Response Toolkit*, and *Integration Platform*. Figure 2 shows the interaction among stakeholders and sub-systems.

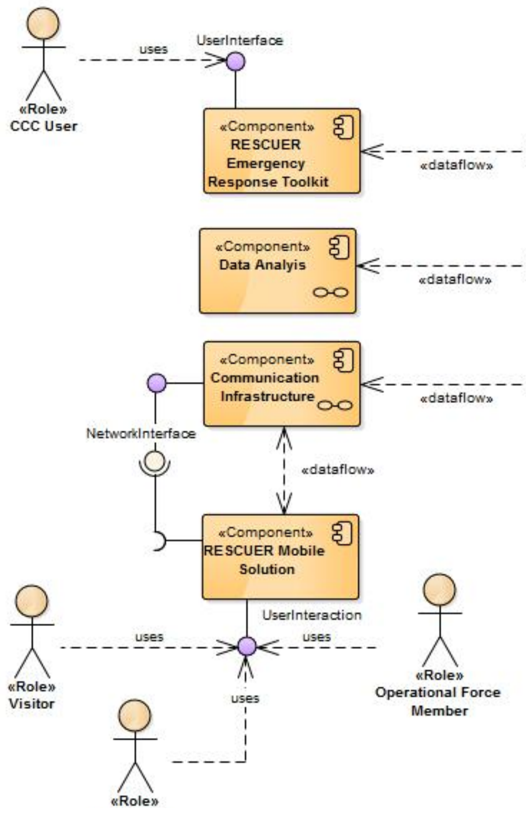


Figure 2. RESCUER sub-systems perspective

The *Mobile Solution* sub-system explores the use of mobile devices to gather information from the crowd in an emergency situation and to support follow-up interactions in an optimized and context-sensitive way. The resulting mobile application interacts appropriately to avoid cognitive overload for the users and to get people engaged in using the RESCUER system.

Aiming to support the information flow between the crowd and the command center, the *Communication Infrastructure* includes a server for receiving, synchronizing, organizing, and storing crowdsourced data from the users' mobile devices. In addition, this sub-system provides a solution for delivering messages to the users' mobile devices in a personalized, location- and situation-sensitive way. This sub-system is also responsible for providing peer-to-peer communication by using the built-in Wi-Fi capability of mobile devices if no Internet connection is available during the emergency.

Automatic data analysis is especially relevant for emergencies in large-event scenarios, where emergency reports from thousands of people are sent to the command center. The *Data Analysis* sub-system receives data to be fused and filtered in order to obtain an enriched collection of data about the emergency situation and thereby enable a more robust and efficient analysis.

The *Emergency Response Toolkit* supports decision-making, coordination of responses, and communication with stakeholders. This sub-system provides appropriate data visualization mechanisms through an intuitive, concise, but resourceful dashboard with modern solutions to map an incident scenario. It also includes a semi-automated solution for communication with the community to provide timely, coordinated, and accurate information about the nature and status of an emergency situation.

In order to assure consistent and efficient interaction among the other sub-systems, the *Integration Platform* provides a communication protocol, storage, and technology to handle message (called topics) exchanges. In general, the sub-systems interact by publishing and subscribing appropriate topics to the *Integration Platform*, which enables normal system execution.

2.2. Distributed Development Setting

The RESCUER project fosters cooperation among companies, research institutions, and universities from Brazil, Germany, and Spain. In addition, this project has partners from industrial parks in Brazil and Austria to validate the proposed solution in a real-world scenario. Software development is being performed by the following members: MTM, DFKI, VOMATEC, Universidad Politécnica de Madrid (UPM), University of São Paulo (USP), and Federal University of Bahia (UFBA).

MTM is a Brazilian company responsible for implementing the *Mobile Solution*, which should run on both the Android and iOS mobile platforms. MTM needs to integrate the mobile solution with the two libraries provided by DFKI, namely the Ad-hoc P2P Library and the Sensor Recording Library. MTM is also responsible for building the BLOB Storage Service that is responsible for storing multimedia data.

DFKI is a German research institute responsible for developing five modules: Ad-hoc Network, Sensor Recorder, Sensor Data Receiver, Sensor Analysis, and SMS Receiver. All these solutions have to be integrated with the *Integration Platform*. In addition, DFKI might need to take care of the User-Interaction Data Receiver and Data Sender components. VOMATEC is a German company responsible for building the *Emergency Response Toolkit*, Combined Analysis Module, and *Integration Platform*.

UPM and USP are Spanish and Brazilian universities, respectively, developing the *Data Analysis* sub-system. UPM is a Spanish university responsible for building the Video Analysis Module interacting with the *Integration Platform*. USP is a Brazilian university responsible for building the Image Analysis Module, which communicates to the *Integration Platform*.

The Fraunhofer Institute for Experimental Software Engineering (Fraunhofer IESE) develops innovative methods and solutions for the development of high-quality, complex information systems and embedded systems. Fraunhofer is responsible for dealing with infrastructure tasks such as requirements engineering, software architecture description, system integration and user interface design.

UFBA is a Brazilian university responsible for building connectors to social media and external legacy systems of the workforces. This university also provides support for developing several modules in the *Emergency Response Toolkit* and *Integration Platform* sub-systems. UFBA is also handling infrastructure tasks.

3. Architectural Challenges

Considering the distributed development scenario abovementioned, the software architecture team faced a set of challenges concerning architecture specification/dissemination and the feasibility of design decisions. The challenges are classified as deployment viewpoint, system performance and context-awareness challenges.

We extend the report of the challenges and solution ideas from Vale et al. (2015), focusing on challenges concerning distributed software development and context-awareness. At the end of this Section, we relate the architectural challenges and the solution ideas applied to mitigate them.

3.1. Deployment Viewpoint Challenges

Deployment viewpoint challenges are related to the construction of a running environment to test and operate the RESCUER system. It involves design decisions such as component deployment by the project partners and the integration of the individual components to generate a unique and transparent system of systems.

- AC01 – *Commercial deployment*: this project intends to develop an experimental crowdsourcing system, bringing several skills together to deliver a proof of concept for the emergency and crisis management domain. RESCUER provides innovative features such as crowdsourcing information gathering, image analysis and video analysis for this specific domain. In this context, it is reasonable starting with moderate expectations from the audience.

  However, the expectation of potential users (professionals from industrial parks in Camaçari-Bahia-Brazil and Linz-Austria) is to use RESCUER in a real-world scenario since its first release. For architectural matters it presents some challenges concerning how to adapt RESCUER and operate it in very different contexts, considering different requirements, laws and regulations. As consequence, the software architecture has to cover several aspects: realizing the variability between different real-world scenarios by achieving component adaptability and negotiating it with the project partners, dealing with different priorities of architectural significant requirements (e.g. availability, performance and robustness) and providing a minimal infrastructure to deploy each RESCUER component.

  Currently, the experimental scenario does not consider any context variations. This is a consequence of our iterative development approach, since the first increments of the architectural views do not fully address commercial expectations. Additionally, many system components face state-of-the-art issues during implementation. As result, RESCUER architects provide documentation with limited features (smaller scope of functionalities) from the expected commercial product;

- AC02 – *Test deployment*: the project is performing an integration taskforce to set up a suitable environment for system and integration testing. To achieve it, the taskforce members (including the architecture team) must negotiate with RESCUER partners to provide the software deliverable and specify the requirements to install it in a controlled setting. The architecture team has to extract the needed information from each partner and document it properly.

  Currently, the architects face the challenge of extracting from the development teams their understanding about the software requirements as well as their architectural knowledge, aiming to verify whether they have a suitable component for system integration and testing. In case of requirements or architectural misunderstanding from the development teams, the component integration and testing might be compromised.

3.2. System Performance Challenges

Performance challenges are related to architectural significant requirements describing any aspect that impacts on the overall system performance.

- AC03 – *Performance from data extraction to visualization*: the time required for gathering crowdsourcing information from the mobile applications, performing data analysis and visualizing the results has to meet the maximum response time of 0.5 seconds. Despite the architecture team has proposed a set of design decisions to address this significant requirement, evaluating it still remains a challenge.

  Architecturally, such an action involves components from different partners around the world. Measuring performance of individual components is trivial, however, evaluating the performance of the end-to-end communication of this scenario implies a consistent system testing environment as well as suitable metrics for this context. Consequently, the architecture has to provide metrics support the configuration of a testing environment in a proper way;

- AC04 – *Scalability*: fostering the loose coupling of architectural components to keep an independent work for the distributed teams comes at a price. There is an architecture significant requirement stating RESCUER must address scalability for 100-200 test users. The partners are spending effort to meet a set of quality attributes considering the components they are developing.

  However, it does not guarantee the RESCUER system achieves the required scalability automatically. The architecture team must provide proper environment and evaluation strategies for this scenario. Once the results do not meet the significant requirements, the architects have to adapt the components description if necessary;

- AC05 – *Extensibility towards new components*: the design decisions addressing independent components in a distributed development setting deal with high coupling. As new requirements arise, RESCUER becomes more complex and their components tend to incorporate more features. It turns fine-grained components into coarse-grained ones.

  In this scenario, the maintainability is compromised and designing smaller/extensible components is essential. In addition, procedures and technologies to provide a comprehensive integration are essential.

3.3. Context-Awareness Challenges

Depending on the type of incident (e.g. fire, explosion or gas leakage), the source of information (e.g. firefighter, affected person or civilian), regulations/laws of the affected area and other aspects, an emergency and crisis management system must behave differently to provide consistent information and combat the incident. Aiming to adapt the system according to several possibilities of an emergency and crisis scenario, the architects concluded RESCUER has to address context-awareness. The context-aware architectural challenges are discussed next.

- AC06 – *Identification of contexts*: understanding the variations of context faced by the RESCUER system is necessary. It will demand requirements elicitation with the potential users (professionals from industrial parks in Brazil and Austria), and the architecture should accommodate it in a self-adaptive system. The current architecture does not support context-aware features. It requires refactoring and all partners must be aware of eventual modifications;
- AC07 – *Context-aware data analysis*: this is the core feature to turn RESCUER into a context-aware system. RESCUER must be able to get raw data (text, images and videos) and generate proper information considering the current context of an incident. It involves a well-defined structure of knowledge about the variations of the context, a reasoning engine to process data and provide context-aware information, and a database to store previous experiences that might help to understand future contexts;
- AC08 – *Context-aware visualization*: the architecture specification describes usability as significant requirement. Visualization mechanisms for context-aware systems is still a research gap, however, RESCUER must adopt existing visualization metaphors and techniques to provide a consistent visualization for mobile and command center users. Considering context-aware visualization is a research topic, the architects must describe a flexible component able to incorporate greater changes requested by the new requirements that can arise for this task.

4. Solution Ideas

Facing the challenges previously presented, the architecture team is incorporating six preliminary solution ideas into the project:

- *S01 (related to AC02, AC03 and AC04) – DevOps toolchain*: incorporating DevOps [Bass et al. 2015] concepts would greatly help in our scenario. It would facilitate continuous integration, continuous testing, and continuous delivery with appropriate tool support. In addition, such tools have features to improve the distributed software development, making the activities and results sharing easier;
- *S02 (related to AC01, AC03, AC04 and AC06) – Continuous feedback from potential end-users*: as the domain is not yet well understood, it is not possible to elicit all requirements in the first attempt. Therefore, it is important to continuously show the prototypes or partially implemented system to the end-users. Refining the requirements and making the system acceptable to the community is not possible without continuous feedback;
- *S03 (related to AC01, AC02, AC03 and AC04) – Plan for creating development/test environment*: it is important for distributed development projects to plan for IT provisioning support. The architects are creating a schedule for system integration and testing, defining which features each component must deliver and the related test cases. Consequently, the

development teams must manage their tasks to achieve the integration goals. It is essential architects to perform such plan, since they have an overall understanding of the RESCUER system. Individual development environments provided by each team are not enough for doing sound integration and acceptance testing;

- S04 *(related to AC05) – Decouple components*: every component needs to be decoupled as much as possible from the rest of the system. A component should provide specific services but it should not be aware of how the service will be consumed;
- S05 *(related to AC03 and AC04) – Continuous testing*: to learn about the current status of the development, it is important to set up an environment and a process for continuous integration. This will make pain-point explicit to everyone;
- S06 *(related to AC06, AC07 and AC08) – Context interpreter component*: this kind of system needs to use modern analysis and visualization techniques. To do an effective and efficient analysis, it is important to know the context of the current state of the system. It is therefore important to build a context interpreter that keeps track of the context and supports all other components with the context information whenever necessary. The whole system should be built around this new component.

5. Conclusion

In this paper, we presented the challenges we experiences in the RESCUER project. Understanding architectural challenges is the key for making any software-intensive system successful. The architectural challenges drive the architecture and it is located at the center of any system development. We classified the challenges according to architectural viewpoints, and we also presented high-level solution concepts, relating them corresponding to each challenge.

We assume this work can benefit other organizations to build such a system in a distributed setting, preparing themselves with respect to the challenges and solutions described here and thus greatly reduce the cost and effort for development. In addition, the architectural challenges can be addressed by further research. As a future activity, we will continue refining the solution concepts and applying them in the RESCUER setting, striving it to obtain evidence with respect to our ideas.

References

Vale T., Arif T., Pedraza L., Vieira V. (2015) “Architecting Crowdsourcing Systems: Challenges and Solution Ideas from the RESCUER Project”. Anais do Primeiro Workshop sobre Sistemas de Crowdsourcing. Belo Horizonte, Brazil.

Bass L., Weber I., Zhu. L. (2015) DevOps - A Software Architect’s Perspective. Pearson Education, Inc.

Thorsten K., Jens K., Matthias, N. (2011) “Architecture-centric software and systems engineering. Fraunhofer ACES: Architecture-Centric Engineering Solutions, IESE-Report, 079.11/E, 2011”, http://publica.fraunhofer.de/dokumente/N-186361.html.

Towards Architectural Synthesis of Systems-of-Systems

Marcelo B. Gonçalves[1,2], **Flavio Oquendo**[2], **Elisa Y. Nakagawa**[1]

[1]ICMC, University of São Paulo, São Carlos, Brazil
[2]IRISA-UMR CNRS/Université de Bretagne-Sud, Vannes, France

{marcelob,elisa}@icmc.usp.br, flavio.oquendo@irisa.fr

Abstract. *A System-of-Systems (SoS) is the result of constituent systems interoperating to achieve common goals. This emerging class of systems brings new development challenges, especially for the design of their software architectures. Despite architectural synthesis is a relevant activity in the architectural process with impact in the quality of software architectures, current software engineering approaches do not properly address it in the context of SoS. To address this lack, the main contribution of this work is to present SASI (SoS Architectural SynthesIs), a method to support architectural synthesis in SoS software architectures. Besides introducing the main elements of SASI, this paper reports results of a study aimed to assess the feasibility of the proposed method.*

Resumo. *Um Sistema-de-Sistemas resulta de sistemas constituintes interoperando para alcançar objetivos comuns. Essa nova classe de sistemas traz desafios de desenvolvimento, especialmente em seu design arquitetural. A despeito da síntese arquitetural ser uma atividade importante no processo arquitetural, com impacto na qualidade das arquiteturas resultantes, as abordagens atuais não tratam adequadamente essa atividade no contexto de sistema-de-sistemas. A fim de tratar essa lacuna, a contribuição principal deste trabalho é a introdução do SASI, um método para oferecer suporte à síntese arquitetural em arquiteturas de software de sistemas-de-sistemas. Além de introduzir os principais elementos dessa proposta, este artigo reporta resultados de um estudo de viabilidade realizado para o método proposto.*

1. Introduction

In the last years, it has been possible to notice an increasing interest in the research and development of *Systems-of-Systems* (SoS), a class of complex systems that stems from the interaction among distributed, heterogeneous, and independent constituent systems that interoperate to form a larger, more complex system for accomplishing global missions [Maier 1998]. The result of such a collaborative work is said to be more than the sum of the constituent systems as it enables the SoS to offer new functionalities, i.e., an emergent behavior that cannot be provided by these constituents individually [Boardman and Sauser 2006].

SoS have followed the natural trend of complex, large-scale systems becoming more software-dependent, leading to the so-called *software-intensive SoS*[1], i.e., SoS in which software plays an essential role in their design, development, and evolution [Gonçalves et al. 2014]. As for any software-intensive system, *software architectures*

[1]For the sake of simplicity, software-intensive SoS are hereinafter referred as SoS.

have been regarded as a significant element for determining the success of such systems and taming their complexity, while contributing to the achievement of important quality requirements such as interoperability and performance. In the SoS context, software architectures must be able to address the inherent characteristics of SoS, encompass the organization of constituent systems, and deal with a dynamic and evolutionary context.

Within the construction of software architectures, the *architectural synthesis* aims to design architectural solutions in order to meet the requirements upon the system that directly influence their architecture, the so-called architecturally significant requirements [Hofmeister et al. 2007]. Despite the relevance of this step as a driver for the system implementation, existing approaches in the literature do not properly address architectural synthesis in the SoS context, being typically concerned with systems engineering and high-level aspects of SoS architectural development. In addition, it is necessary to reason which are the fundamental elements for the architectural synthesis in the development of SoS software architectures.

Aiming at tackling these issues, this paper presents SASI (SoS Architectural Synthesis), a method for establishing and managing architectural solutions in *acknowledged SoS*, in which goals, resources, and authority are all recognized at SoS level, but the constituent systems retain their independent management and the behavior is not subordinated to the central managed purpose [DoD 2008]. SASI was conceived to provide guidelines on what must be done in the construction of acknowledged SoS software architectures and support how to perform the required activities for architectural synthesis in any application domain. In order to assess the feasibility of our proposal, we conducted an observational study with six participants. The obtained results contributed to improve SASI and to verify its applicability and independence in terms of application domain.

The remainder of this paper is structured as follows. Section 2 introduces the issues related to the architectural synthesis of SoS software architectures. Section 3 presents our proposal to support this activity. Section 4 presents the evaluation concerning our proposal. Finally, Section 5 outlines some conclusions and directions to future work.

2. Architectural Synthesis of SoS

According to the general model of architectural design proposed by Hofmeister et al. [Hofmeister et al. 2007], the major activities on building any software architecture are (i) architectural analysis, (ii) architectural synthesis, and (iii) architectural evaluation. In the architectural analysis, architecturally significant requirements are elicited expressing which problems in the system context the software architecture can solve. In turn, the architectural synthesis encompasses the design and proposition of candidate architectural solutions in order to effectively solve the architecturally significant requirements, moving from the problem to the solution space. Finally, the architectural evaluation aims at evaluating the proposed architectural solutions against the architecturally significant requirements. In order to verify if the design decisions made are the right ones.

Despite the existence of approaches covering the synthesis of software architectures, the class of SoS reaches a complexity threshold in which traditional software engineering is no longer sufficiently adequate [Boehm and Lane 2006]. Due the inherent characteristics of SoS, they software architecture differ from a monolithic systems on several issues, such as the communication involving multiple stakeholders and organizations, evolutionary development, dynamism in an operational environment based on emergent behaviors [DoD 2008]. As a result, SoS software architectures have been constructed through ad-hoc perspectives in which each SoS has its software architecture developed on a particular manner. Therefore, there is a lack of further investigations on the systematic development of SoS software architectures (and more specifically on the architectural synthesis) in order to improve the available solutions to support the architectural development of these architectures.

Due the independence of constituent systems, SoS software architectures are inherently dynamic. Moreover, emergent behaviors result from the collaborative work of constituent systems and these systems are not totally subordinated to SoS interests. Therefore, the architectural solutions must consider the relevance of self-organization concerns and prediction of both desired and undesired emergent behaviors. In general, desirable behaviors come from architectural solutions and must be maximized since they foster the accomplishment of SoS missions. On the other hand, undesirable behaviors must be minimized because they may negatively affect the accomplishment of SoS missions and/or important quality attributes such as performance, security, and reliability.

3. SASI: A Method for Architectural Synthesis of SoS

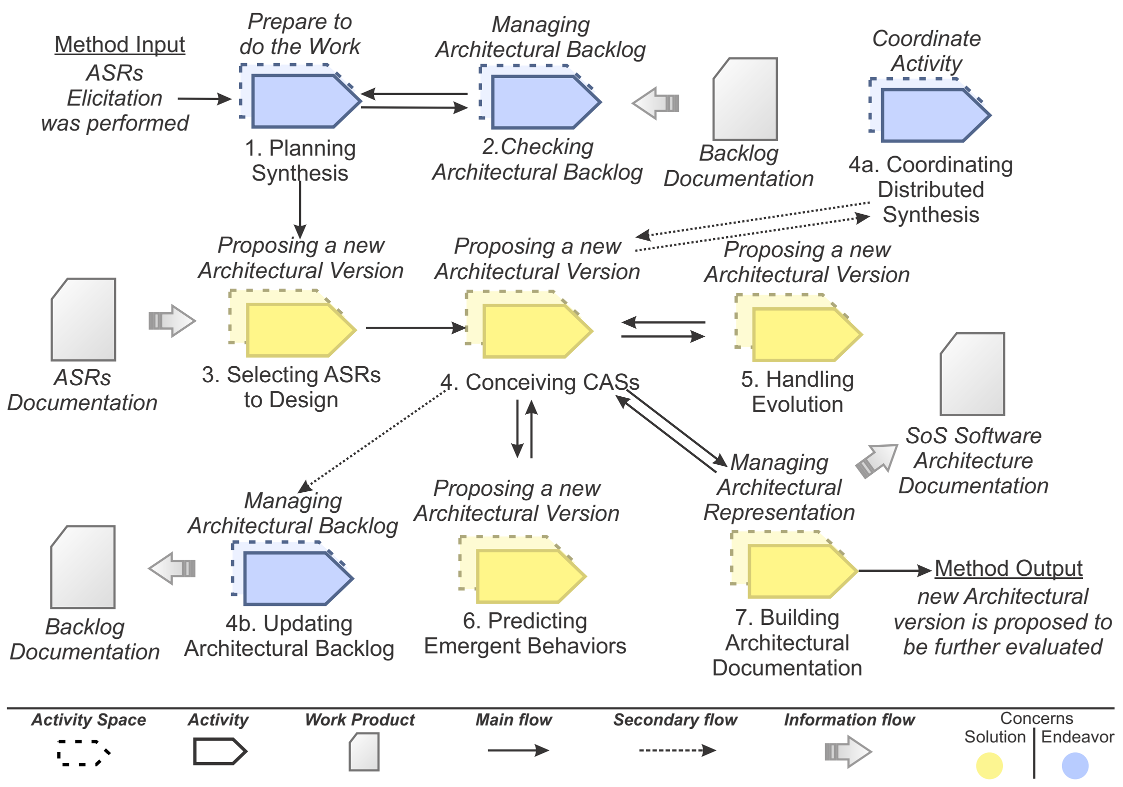
SASI, our method to support architectural synthesis of acknowledged SoS, is structured upon the OMG's Essence Standard[2], which comprises a language for method authoring and a kernel designed to be a reference for software development projects [Jacobson et al. 2013]. With the Essence Language, it is possible to instantiate processes by using *kernels* and *practices*: kernels provide conceptual grounding and guidelines to "what must be done" whereas practices provide elements such as specific activities and work products determining the "how to do". Our previous work [Gonçalves et al. 2015] proposed the SoS SA Kernel, a kernel that adheres to the Essence Kernel for determining the "what must be done" in the construction of acknowledged SoS software architectures, independently from application domains, work products, and organizational contexts. We have taken advantage of SoS SA Kernel when conceiving SASI in order to support project teams on the instantiation or improvement of their own development processes targeting acknowledged SoS.

The main elements borrowed from the Essence Language and employed in the representation of SASI are *alphas*, *work products*, *activity spaces*, and *activities*. Alphas determine the "things to work with" of methods and have a set of progression states verified through checklists. Changes in these states indicate work progress and can help development teams to understand their own way of working. A work product is an artifact that concretely represents an alpha, e.g., a document or a code slice. Activity Spaces determine the "what must be done" in development projects and activities define approaches to accomplish activity spaces by providing guidelines on how to work with alphas when following a given method. Figure 1 shows the main workflow of SASI for architectural synthesis. Next subsections detail the main elements of the method.

3.1. SASI Work Products

In our previous work [Gonçalves et al. 2015], several alphas were established in order to encompass all the "things to work with" when architecting acknowledged SoS. Follow-

[2]Essence Standard is avaiable at <`http://www.omg.org/spec/Essence/`>



Method Input
ASRs Elicitation was performed
Prepare to do the Work
1. Planning Synthesis
Managing Architectural Backlog
2.Checking Architectural Backlog
Backlog Documentation
Coordinate Activity
4a. Coordinating Distributed Synthesis
Proposing a new Architectural Version
ASRs Documentation
3. Selecting ASRs to Design
4. Conceiving CASs
5. Handling Evolution
SoS Software Architecture Documentation
Managing Architectural Representation
Managing Architectural Backlog
4b. Updating Architectural Backlog
6. Predicting Emergent Behaviors
7. Building Architectural Documentation
Method Output
new Architectural version is proposed to be further evaluated
Activity Space
Activity
Work Product
Main flow
Secondary flow
Information flow
Concerns
Solution
Endeavor

Figure 1. SASI workflow.

ing, we describe the work products introduced by SASI that might express these alphas considering only the perspective of architectural synthesis.

Architectural Backlog Documentation. The Architectural Backlog encompasses any relevant matter related to the architecture and it can be updated at any time in the life cycle. In this context, the Architectural Backlog documentation must comprise the knowledge to guide the architectural development, including issues such as the prioritization of architectural requirements, feedback of problems found during the architectural evaluation, and registration of new ideas for viability analysis in further iterations.

Architecturally Significant Requirements (ASRs) Documentation. An ASR is any functional or non-functional requirement relevant for the SoS software architecture and therefore drives the architectural design. In the SoS context, ASRs are often related to quality attributes, missions, constraints, and requirements derived from environmental conditions. As a recommended work product, the ASRs documentation is obtained after agreement through different stakeholders to be further handled by the architecture, including, for example, a quality model created to provide specific metrics and constraints for relevant quality attributes in SoS such as interoperability, security, and performance. Furthermore, despite we do not expect all ASRs to be known a priori, the ASRs elicitation is a precondition to perform architectural synthesis.

SoS Software Architecture Documentation. The architectural documentation is proposed to explicity express, support evaluation, and convey an SoS software architecture. Therefore, it must provide elements required to adequately express the architecture of the system, such as architectural viewpoints and views, or architectural prototypes. Moreover, it can be made with different representation techniques (informal, semi-formal, and formal) and must cover the different views (e.g., structural and behavioral) to provide a better understanding of the system architecture to both stakeholders and developers.

3.2. SASI Activities

Following, we present the activities defined in SASI.

1. Planning Synthesis. This activity deals with the establishment/update of planning issues concerning the execution of synthesis activities. It includes scheduling time and resources, e.g., the establishment and management of the team and its way of working with stakeholders and other project teams, as well as the review of previous iterations in order to maintain the architectural design as expected.

2. Checking Architectural Backlog. This activity refers to the strategic checking of the Architectural Backlog, which can be used as information source to support architectural activities.

3. Selecting ASRs to Design. This activity is about the decision and establishment of what must be designed in each synthesis iteration based on the set of ASRs to be handled. Several factors can influence the decision about the how many ASRs will be included, such as the diversity of application domains, the architects expertise in these domains, and the teams size.

4. Conceiving Candidate Architectural Solutions (CASs). In this activity, a set of CASs is proposed to encompass the ASRs under design, thus meeting a set of ASRs and establishing a new architectural version to be further evaluated. The CASs can be created based on several different sources, such as a background on building similar architectures, frameworks, design checklists, domain decomposition, reference architectures, and architectural patterns [Bass et al. 2012].

5. Coordinating Distributed Synthesis. SoS development typically involves different organizations performing a collaborative, distributed development of the software architecture. This activity space must encompass the required support for such a collaborative work through heterogeneous teams. In this sense, it encompasses the management of the distributed work to be performed as expected, e.g., negotiation of authority levels and communication strategies.

6. Updating Architectural Backlog. This activity space encompasses registering and maintaining inter-dependencies, trade-offs, changes, traceability links, as well as any other relevant information regarding the performed synthesis.

7. Handling Evolution. Evolvability is the ability to easily accommodate future changes. This attribute is highly required in SoS scenarios since the architecture is constantly evolving. In this context, this activity determines the investigation and establishment of strategies to maintain the SoS evolvability. Since CASs must be established by considering the evolvability concern, this activity dialogues with the *Conceiving CASs* influencing in the proposed architectural version.

8. Predicting Emergent Behaviors: Emergent behaviors of an SoS are not simply a sum of parts (i.e., constituent systems and their capabilities) and an effort must be directed trying to predict it also considering the identification of both desired and undesired behaviors. Despite the management of these behaviors includes other phases of architectural construction (i.e., architectural analysis and evaluation), it is possible to conduct efforts in the architectural synthesis on how the proposed CASs can promote desired behaviors and also the analysis of how behaviors from previous iterations interfere in these CASs.

9. Building Architectural Documentation. The proposed CASs must be reflected in assets that allow the further understanding and evaluation of what was proposed. In this activity, the SoS software architecture is described according to the development context of each SoS, thus considering different interests, viewpoints, and particular environments. Moreover, it must provide a documentation that reflects the current SoS software architecture and also a way to convey the SoS software architecture that reaches all interested stakeholders and developers.

4. Evaluation

Aiming to assess the feasibility of applying SASI to generate method instances for SoS in any application domain, we conducted a study with six graduate students from the University of São Paulo during the Fall 2015 semester. A secondary goal was to receive feedback on the format and contents of the SASI description, which was used to make enhancements in SASI. In this context, the study has concentrated on the identification of SASI elements to conceive a method instance for a particular SoS context and the ability of understanding such elements on any SoS application domain.

The materials used during this study consisted of (i) the initial version of SASI description in the Essence Language, (ii) descriptions of two SoS inn different application domains, and (iii) a form to be filled by the subjects. Participants first received a training on SASI and then they were grouped into two teams. Each team received a different system description and each subject tried to compose a method instance by identifying which SASI elements (activities, alphas and work products) would be required to establish a method for architectural synthesis considering the provided SoS description. Moreover, the subjects had to justify why not included a given activity and to point out the alpha states before and after the architectural synthesis.

In order to verify if the method instances were as expected, they were compared to templates including all the adequate SASI elements on each SoS description. Figure 2 summarizes results obtained with the observational study. The quantitative data from this study showed some positive results, we observed that the instances generated by the subjects achieved a high degree of conformance (i.e., all evaluation averages reached at least 65% of conformance) to the templates and the conformance reached by the two teams was similar. In this context, we concluded that it was possible to generate instances of architectural synthesis methods on specific development contexts by using SASI, in spite of its generality regarding application domains.

Qualitative data also provided us with some lessons to enhance SASI. Subjects were questioned about their personal usage experience with SASI and asked for providing additional comments and enhancement suggestions. The main suggestions provided by the experts and further incorporated into SASI were, (i) more details about the relationships among activities and (ii) offering a template for the Architectural Backlog and

ASRs documentation as means of enhancing the guidelines concerning which information/asset should be registered/updated. In general, we consider that the results of this study represent a good indicative that SASI can be an adequate, comprehensive method for supporting architectural synthesis in acknowledged SoS software architectures.

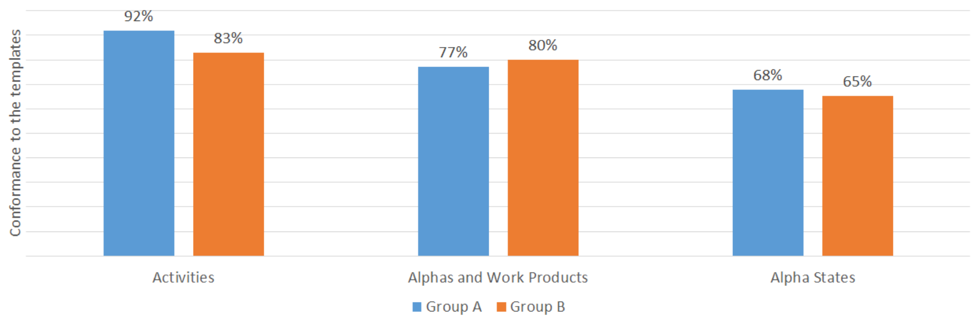


Figure 2. Conformace with templates.

4.1. Threats to Validity

The conducted study and its results may have been affected by some threats to empirical validity. Following, we briefly discuss some of these limitations.

Internal Validity. Internal validity is mainly concerned with unknown factors that may influence the results. To increase the validity of our study regarding this concern, we carefully designed, piloted, and iteratively refined the form and documentation provided to the subjects. Additionally, we made the participation voluntary and anonymous.

External Validity. External validity is related to claims for the generality of the presented results. In this context, we believe that the number of participants can be accepted since our main goal was to observe the results of using SASI and gain insights and suggestions for improving it.

Construct Validity. Construct validity focuses on the correct interpretation and measurement of the perceptions, i.e., the relationship between concepts and theories behind the study and what is actually measured and affected. We attempted to mitigate most of bias coming from the subjects by structuring a significant part of the form to drive the use of SASI in a particular SoS context. Furthermore, since enhancement suggestions could yield different interpretations, answers to these questions and consequent improvements were discussed among the researchers.

5. Conclusions

Despite the growing presence of SoS in several application domains and societal sectors, there are no standard processes or consensual practices regarding the construction of SoS software architectures. For this reason, important investigations still remain not addressed, such as the architectural synthesis on SoS software architectures. To fulfill this gap, the main contribution of this work is to propose a method to support this activity in acknowledged SoS context. We evaluated the feasibility of SASI in an study which shown good results on generating specific method instances with SASI.

Taking in account the relevance of adequate synthesis for architectural construction in SoS scenarios, future work will encompass a deeper investigation on the application of SASI in industry. In this context, we also intend to investigate and propose methods for the other major activities of architectural design, i.e., architectural analysis and architectural evaluation.

References

Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in practice*. Addison-Wesley, USA, 3rd edition.

Boardman, J. and Sauser, B. (2006). System of systems – The meaning of *of*. In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, Piscataway, NJ, USA. IEEE.

Boehm, B. and Lane, J. (2006). 21st century processes for acquiring 21st century software-intensive systems of systems. *Journal of Defense Software Engineering*, 19(5):4–9.

DoD (2008). *Systems Engineering Guide for Systems of Systems*. Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, Washington, DC, USA. Version 1.0.

Gonçalves, M. B., Cavalcante, E., Batista, T., Oquendo, F., and Nakagawa, E. Y. (2014). Towards a conceptual model for software-intensive system-of-systems. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 1605–1610, Washington, DC, USA. IEEE Computer Society.

Gonçalves, M. B., Oquendo, F., and Nakagawa, E. Y. (2015). A meta-process to construct SoS software architectures. In *Proceedings of the 30th ACM/SIGAPP Symposium on Applied Computing*, pages 1411–1416, New York, NY, USA. ACM.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., and America, P. (2007). A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126.

Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., and Lidman, S. (2013). *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

A Biological Inspiration to Support Emergent Behavior in Systems-of-Systems Development

Valdemar Vicente Graciano Neto[1,2], **Elisa Yumi Nakagawa**[2]

[1]Instituto de Informática (INF/UFG), Universidade Federal de Goiás, Goiânia, Brazil

[2]ICMC, University of São Paulo, São Carlos, Brazil

valdemarneto@inf.ufg.br, elisa@icmc.usp.br

Abstract. *Systems-of-Systems (SoS) are engineered with pre-existing software called constituents. SoS development requires a new software engineering endeavor to address the SoS' particularities. Computer science has a tradition in looking for inspiration in biological systems to propose solutions to solve complex problems. Remarkable examples include solutions inspired in ant colonies, swarm of bees, and neural networks. Recent results have been communicated regarding a living cell software-based simulation. In that simulation, the entire life cycle of a living cell is simulated as a function of individual capabilities provided by individual software modules. Those modules simulate the behavior of inner structures of a cell. For cells (or simulated cells) and for SoS, independent parts contribute to deliver a more complex behavior. Then, we investigated how the similarities between SoS and cells structure, and results from cell simulation could foster SoS engineering. This paper presents preliminary results of this research, reporting perceptions we realized between those two research topics, as well as some insights on how both areas could cross-fertilize each other.*

1. Introduction

Systems-of-Systems (SoS) are software-intensive systems constructed with pre-existing systems called constituents. Those constituents, when working together, can deliver complex behaviors that they could not exhibit if working in isolate. Despite the existence of successful proposals to integrate such constituents into an operational SoS [Nakagawa et al. 2014], Software Engineering community has investigated best approaches, practices, methods, models, and processes to suitably address a forthcoming class of civil systems [Graciano Neto et al. 2014]. Some examples include Smart Cities, Smart Buildings, Smart Grids, and all sort of smart systems composed by other systems and delivering innovative complex functionalities. On the other hand, bio-inspired solutions have been explored in the last years for software development initiatives purposes, such as ant colonies, swarm of bees, neural networks [Dorigo and Birattari 2010, Karaboga et al. 2014, Schmidhuber 2015], and even software ecosystems [Santos et al. 2014b, Jansen and Cusumano 2012], which are inspired in real ecosystems.

In this direction, recent results reported a successful simulation of an alive cell. Covert et al. have developed a software-based simulation that integrates several independent software modules to emerge a complex emergent functionality: a simulation of a complete life cycle of an alive cell [Karr et al. 2012, Covert 2014, Matsuoka and Shimizu 2015]. Since simulation is a recurrent paradigm in Systems Engineering guides [Graciano Neto et al. 2014], and suitable models are still necessary to represent the whole dynamics of large complex smart SoS, we envisioned a possibility of migrating those concepts and results presented by Covert et al. to open a new possible research path regarding SoS development: a bio-inspired SoS development. We understand that this research topic can be plausible, since other successful bio-inspired initiatives are widespread in Computer Science.

This paper presents some insights about a cross-fertilization that could be explored regarding a bio-inspired approach to support SoS development. We expect that these preliminary discussions can open a new research branch in SoS development, enabling prominent results from a multidisciplinary research perspective. Remainder of this paper is structured as follows: Section 2 brings the necessary background to start the discussion, Section 3 presents the first insights we gathered, Section 4 discusses our preliminary findings, and Section 5 concludes the paper, presenting remarks and future research.

2. Foundations

Systems-of-Systems (SoS) are an arrangement of software systems called constituents. Such constituents are pre-existing independent software which contribute with their individual functionalities for accomplishing innovative SoS purposes [Boardman and Sauser 2006]. SoS are engineered to articulate their constituents to offer higher-level functionalities [Nakagawa et al. 2014, Santos et al. 2014a]. Those functionalities emerge as a synergistic result of constituents' interoperability. SoS are required to accomplish missions, a higher goal structured as a set of tasks to be performed by the constituent systems. Constituents might be unaware of their cooperation and information exchanging in order to accomplish it. Each constituent system accomplishes its own individual mission and is able to contribute to the accomplishment of the global mission of the SoS [Silva et al. 2014].

Translating the SoS concept into reality requires a different approach despite the one currently used to engineer systems [Boardman and Sauser 2006]. Part of such additional complexity is due to SoS' inherent characteristics. SoS are distinguished from large monolithic systems by a set of key 'dimensions' postulated by Maier [Maier 1998, Nielsen et al. 2013, Fitzgerald et al. 2012, Andrews et al. 2013, Pérez et al. 2013]: Operational independence of constituents and Managerial independence of constituents, emergent behavior, and evolutionary development processes (evolution), besides geographical distribution as an essential feature [Board 2014]. Besides Maier's dimensions, dynamic architecture, and self-management aspects [Nielsen et al. 2013, Fitzgerald et al. 2012, Andrews et al. 2013, Weyns and Andersson 2013, Batista 2013, Romay et al. 2013] have been recognized as inherent and expected characteristics for software-intensive SoS.

When considering SoS engineering, it is important to highlight that constituents might be monolithic systems, or even other smaller SoS themselves, delivering specific functionalities which contribute to the larger SoS' mission. Thus, under this perspective, SoS can be constituted by subsequently smaller parts, also considered constituents themselves.

Under another perspective, any product of engineering is based on models, including SoS. Independently of the various modeling approaches available, a proper modeling is useful in a range of areas. Appropriate models helps to realize the systems' behavior and reveal the underlying principles of the target real element that is being modeled [Matsuoka and Shimizu 2015]. With such appropriate models, it is possible to simulate real behaviors, predicting effects, and supporting necessary calibrations, adaptations, and changes which can, in an early stage, avoid errors and losses.

One of the frequent models present in computer science are the bio-inspired models. Among recent initiatives, one is remarkable: a living cell simulation [Karr et al. 2012, Freddolino and Tavazoie 2012, Covert 2014, Matsuoka and Shimizu 2015]. Markus W. Covert, a bioengineer, developed a well-succeeded simulation of a living cell. Authors developed a sort of software models and modules representing, each one, a particular cellular inner process, such as DNA processing, RNA transcription, and cell division, originating a new cell [Karr et al. 2012, Covert 2014]. They developed 28 independent modules which, interacting, offered a complete living cell life cycle as an emergent result. They claim that such interactions might help scientists to understand cause-effect relations among organelles into a real cell.

They selected a quite simple living unicellular bacterium called *Mycoplasma genitalium*, which have only 525 genes against the almost 20 thousand genes a human cell has. Duplication period manifested as an emergent property that results from a complex interaction between distinct replication phases. They developed a comprehensive whole-cell model. Simulations, source code, knowledge database, visualization code, and experimental data are available online[1].

3. A bio-inspired solution for SoS's emergent behavior

Biological analogies are recurrent in computer science. Ant colony [Dorigo and Birattari 2010], swarm of bees [Karaboga et al. 2014], neural networks [Schmidhuber 2015], and genetic algorithms [Karakatic and Podgorelec 2015] are examples of computational solutions whose operating principles are based on existing real biological systems. Thus, given SoS inherent characteristics, and living whole-cell simulation results, features, and open issues, we envisioned some points of intersection between both areas that could benefit each other to join efforts toward unified results.

Briefly, a cell can be individually considered as an SoS. The interaction among their organelles displays behaviors. Together, such interactions and individual behaviors culminate in cellular function and dynamics. In this sense, a biological analogy can help to realize how emerging behaviors occur and how this spontaneous phenomenon can be represented and raised as a function of individual constituents and their individual capabilities. Thus, it can be possible to migrate such interaction strategies among constituents to better 1) represent interactions among constituents, 2) realize the impact of constituents' individual influences over the whole behavior, and 3) engineer such emergent behaviors in SoS.

In fact, the whole human body can be seen as a huge and complex SoS, where its constituents are themselves, other complex SoS. Human body SoS is divided in organs. Organs interact among them to deliver an emergent behavior: your life. Organs are themselves other SoS, since they are constituted by specific tissues. Tissues are also SoS, since they are a composition of cells. And cells themselves are minor SoS constituted by organelles, such as Golgi complex, plasma membrane, and mitochondria. Organelles

[1]`https://simtk.org/home/wholecell`

interaction deliver the whole-cell life cycle as an emergent behavior. Such as automotive systems, which are composed by smaller parts which interact to deliver the car operation, biological systems can be perceived under an SoS perspective.

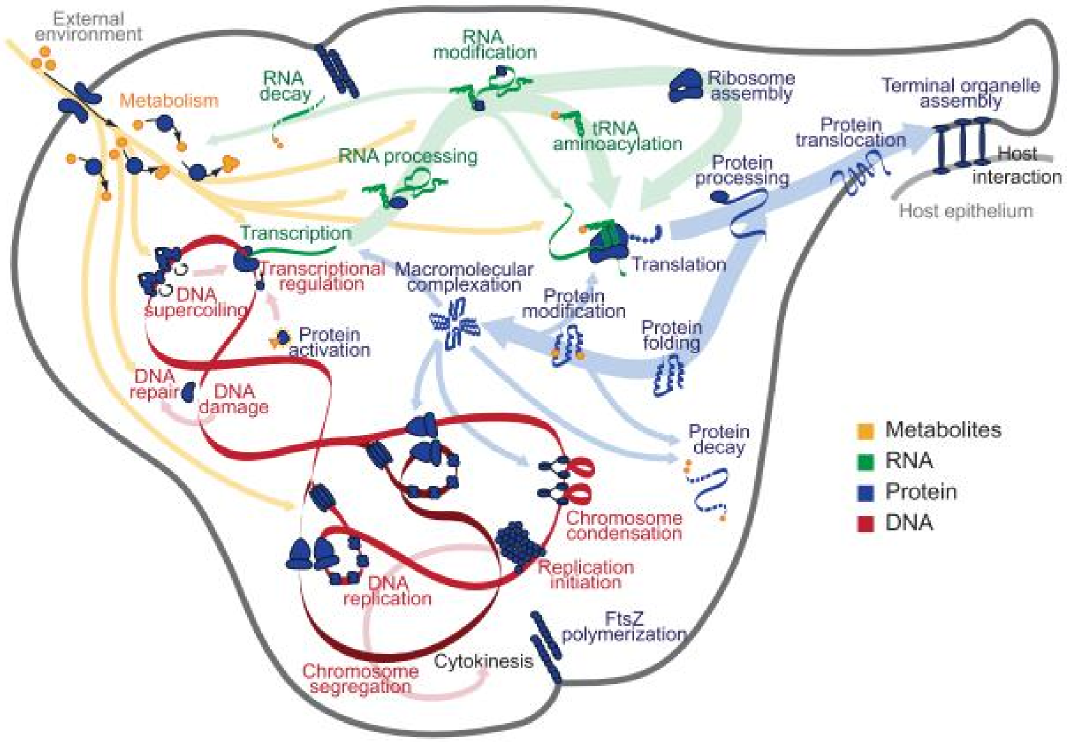


Figure 1. A model of a Cell separated by organelles [Karr et al. 2012].

Figure 1 presents a model of a living cell. That figure helps to realize a cell under an SoS perspective. Under this perspective, many small distinct structures have independent work, such as RNA processing, RNA decay, RNA modification, Ribosome assembly, and Protein processing. Those structures play, each one, a constituent role, delivering individual capabilities which, together, contribute to the whole cell life cycle, until the cell division.

We could establish a comparison between SoS characteristics and a cell (as an alive SoS) characteristics. That comparison is available in Table 1. To establish such comparison, we based it on the criteria available in a conceptual model for SoS available in literature that lists all of the essential characteristics an SoS should exhibit [Benites Goncalves et al. 2014]. Under this perspective, organelles work as constituents of the Cell SoS. The Global Mission could be growing, reproducing, or even the cell entire life cycle. About software dominance, the simulated cell is based on several software modules. Each organelle offers independent behavior, what characterizes operational independence of their constituents. Each organelle is represented by one or more software modules, and those modules are required to evolve to approach their configuration parameters to reliably represent a real cell. Thus, the simulated cell requires evolutionary development. The "execution" can be considered the emergent behavior. Geographic distribution is a relative concept, and could be considered in the following way: if the

SoS	**Cell**
Constituents	Organelles
Global mission	Growing and reproducing
Software dominance	The simulated cell is based on 28 distinct software modules
Operational independence	Each organelle or inner structure (simulated as a software module) has independent behavior and operation
Evolutionary development	Improvements and calibrations in the parameters which rules the organelles or cell structures operation require constant evolution of modules, featuring evolutionary development
Emergent behavior	Cell's metabolism and life cycle are the emergent behaviors
Geographical distribution	The real cell has a minimum distance among their organelles. Simulation dispenses it.
Connectivity (Interoperability)	Also known as interoperability, organelles interact to deliver emergent behavior.

Table 1. Comparison between SoS and cells under Benites et. al perspective [Benites Goncalves et al. 2014].

structures interoperating are "physically separated", they deliver geographic distribution. Since organelles are physically distinct entities, the real cell offers such distribution, and the software modules simulate it. Finally, such organelles interact and are connected through their inner processes to deliver the emergent behavior. Then, they offer interoperability. And, for the set of characteristics they exhibit, a cell can actually be considered a biological SoS.

4. Discussion

We glimpsed both research areas as potential to benefit each other. SoS community can benefit from cell simulation, and cell simulation can benefit from SoS approaches. Cell simulation approach delivers techniques and source code to simulate a living cell. It can be adapted to engineer suitable simulation environments for the forthcoming smart SoS, using such prominent results regarding emergent behaviors as a composition of individual capabilities to an effective emergent behavior in SoS (a still open issue). Conversely, an SoS development approach could benefit whole-cell simulation, providing a lighter approach of simulation for it. Well-succeeded SoS simulation approaches, such as Agent-based SoS simulation approaches [Pavon et al. 2011], can be migrated for cell simulation, improving those results and visualization. This kind of adaptation can bring advantages for the simulator maintainability and development, since it uses an approach already well-succeeded for SoS issues.

Simulation is, in fact, a recurrent and traditional paradigm and a genuine step in Systems and SoS Engineering approaches [Graciano Neto et al. 2014]. It brings important advantages, such as an early perception of errors, defects, and problems that need to be corrected in specification level before integration step being conducted; and better consistency, verification and validation. Simulation may early evidence phenomena if simulation has been faithfully modeled, providing data not previously obtained.

Simulation make intensive use of models. Conversely, models give support for verification and validation processes, through the use of simulation or other automated techniques. Some types of models are called *runtime models*, enables simulation of the SoS operation via model execution. Agents are frequently mentioned as a complete and mature technology to perform SoS simulation, and can be considered the state-of-art for SoS simulation. SiCoSSyS approach is a sample approach to engineer SoS based on agents simulation [Pavon et al. 2011]. In fact, agents could be used as a lightweight approach to represent each of the modules listed by Covert et al., inovating in cell simulation, and providing the cellular structures capabilities individually, and the whole life cycle as an emergent behavior.

On the other hand, the realization of organelles interaction to deliver its life cycle as an emergent behavior could benefit SoS community. Cell structure interaction patterns could be investigated and reproduced in SoS development, achieving an important requirement imposed by SoS engineering initiatives: addressing of emergent behavior. In cells, emergent behavior really *emerge* as a result of the synergy between the parts which interact. However, in SoS, the emergence is deliberately and intentionally designed [Boardman and Sauser 2006]. Boardmand and Sauser mention that emergent behavior dare not be restricted to what can be foreseen or deliberately designed. They claim that an SoS must be richer in emergence, and that a challenge for the SoS designer is to know, or learn how, as the SoS progresses through its series of stable states, to create a climate in which emergence can flourish, and an agility to quickly detect and destroy unintended behaviors, much like the human body deals with unwanted invasions [Boardman and Sauser 2006]. Following this trend, investigating such recent results in cell simulation could benefit SoS engineering to provide and accomplish some highlighted desired challenging characteristic.

Regarding related work, Boardman and Sauser [Boardman and Sauser 2006] also establish some parallels between the concept of software systems and biological systems, such as the similarities between the human brain and their neurons as constituents, and the colonial behavior shared by ants. However, authors do not discuss how those similarities could be explored to benefit both areas, SoS engineering/simulation and biological simulations.

Indeed, e-Science community[2] has approached some efforts to connect biology, astronomy, and other science efforts to computational solutions. e-Science consists of science that is carried out in highly distributed environment, using computationally intensive solutions [Taylor et al. 2014]. We did not find any evidences of an SoS approach for cell simulation or a cell simulation approach for SoS conception being communicated, as in SoS community, as in e-Science community.

5. Final Remarks

This paper presented some insights for a cross-fertilization between living cell simulation and SoS development areas. We offer this initial results as an starting point of investigation for both e-Science and SoS communities. We expect that this first effort can serve

[2]http://www.nesc.ac.uk/

as an enlightening result, which effectively establish an existent parallel among those research areas. Bio-inspired solutions are recurrent in Computer Science. We expect that our envisioned approach, despite the glimpsed possibilities for cell simulation advances itself, could contribute to improve techniques for effectively providing emergent behavior for SoS. Such SoS intrinsic feature is still an open issue. We wish to provide it for SoS not as a mechanical and nondynamic property, but as a genuine and adaptive condition.

6. Acknowledgements

Authors thank Goiás Research Foundation (FAPEG) and FAPESP by financial support under grant number 09/2013, ID 2013.009.97100854, and grant number 2014/02244-7, respectively.

References

Andrews, Z., Payne, R., Romanovsky, A., Didier, A., and Mota, A. (2013). Model-based development of fault tolerant systems of systems. In *SysCon*, pages 356–363.

Batista, T. (2013). Challenges for SoS Architecture Description. In *1st SESoS*, SESoS '13, pages 35–37, New York, NY, USA. ACM.

Benites Goncalves, M., Cavalcante, E., Batista, T., Oquendo, F., and Yumi Nakagawa, E. (2014). Towards a conceptual model for software-intensive system-of-systems. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 1605–1610.

Board, B. E. (2014). The guide to the systems engineering body of knowledge (sebok). Technical report.

Boardman, J. and Sauser, B. (2006). System of systems - the meaning of of. In *System of Systems Engineering, 2006 IEEE/SMC International Conference on*, pages 6 pp.–.

Covert, M. W. (2014). Simulating a Living Cell. *Scientific American*, (310):44–51.

Dorigo, M. and Birattari, M. (2010). Ant colony optimization. In Sammut, C. and Webb, G., editors, *Encyclopedia of Machine Learning*, pages 36–39. Springer US.

Fitzgerald, J., Bryans, J., and Payne, R. (2012). A formal model-based approach to engineering systems-of-systems. In Camarinha-Matos, L. M., Xu, L., and Afsarmanesh, H., editors, *Collaborative Networks in the Internet of Services*, volume 380 of *IFIP AICT*, pages 53–62. Springer Berlin Heidelberg.

Freddolino, P. and Tavazoie, S. (2012). The dawn of virtual cell biology. *Cell*, 150(2):248 – 250.

Graciano Neto, V. V., Guessi, M., Oliveira, L. B. R., Oquendo, F., and Nakagawa, E. Y. (2014). Investigating the model-driven development for systems-of-systems. In *SESoS*, ECSAW '14, pages 22:1–22:8, Vienna, Austria. ACM.

Jansen, S. and Cusumano, M. (2012). M.: Defining software ecosystems: A survey of software platforms and business network governance. In *Proceedings of the international Workshop on Software Ecosystems*, pages 41–58.

Karaboga, D., Gorkemli, B., Ozturk, C., and Karaboga, N. (2014). A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 42(1):21–57.

Karakatic, S. and Podgorelec, V. (2015). A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing*, 27(0):519 – 532.

Karr, J. R., Sanghvi, J. C., Macklin, D. N., Gutschow, M. V., Jacobs, J. M., Bolival, B., Assad-Garcia, N., Glass, J. I., and Covert, M. W. (2012). A Whole-Cell Computational Model Predicts Phenotype from Genotype. *Cell*, 150(2):389–401.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

Matsuoka, Y. and Shimizu, K. (2015). Current status and future perspectives of kinetic modeling for the cell metabolism with incorporation of the metabolic regulation mechanism. *Bioresources and Bioprocessing*, 2(1):4.

Nakagawa, E. Y., Capilla, R., Díaz, F. J., and Oquendo, F. (2014). Towards the dynamic evolution of context-based systems-of-systems. In *WDES 2014*, pages 45–52, Maceió, Brazil.

Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., and Peleska, J. (2013). Model-based engineering of systems of systems. Technical report. Available from http://www.compass-research.eu/resources/sos.pdf.

Pavon, J., Gomez-Sanz, J., and Paredes, A. (2011). The sicossys approach to sos engineering. In *SoSE 2011*, pages 179–184, Irvine, CA, USA.

Pérez, J., Díaz, J., Garbajosa, J., Yagüe, A., Gonzalez, E., and Lopez-Perea, M. (2013). Large-scale smart grids as system of systems. In *SESoS 2013*, pages 38–42, Montpellier, France.

Romay, M. P., Cuesta, C. E., and Fernández-Sanz, L. (2013). On self-adaptation in systems-of-systems. In *1st SESoS*, SESoS '13, pages 29–34, New York, NY, USA. ACM.

Santos, D. S., Oliveira, B., Guessi, M., Oquendo, F., Delamaro, M., and Nakagawa, E. Y. (2014a). Towards the evaluation of system-of-systems software architectures. In *WDES 2014*, pages 53–57, Maceió, Brazil.

Santos, R., Gonçalves, M., Nakagawa, E. Y., and Werner, C. (2014b). On the relations between systems-of-systems and software ecosystems. In *WDES 2014*, pages 58–62, Maceió, Brazil.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61(0):85 – 117.

Silva, E., Cavalcante, E., Batista, T., Oquendo, F., Delicato, F. C., and Pires, P. F. (2014). On the characterization of missions of systems-of-systems. In *Proc. of the ECSAW*, pages 26:1–26:8, Vienna, Austria. ACM.

Taylor, I. J., Deelman, E., Gannon, D. B., and Shields, M. (2014). *Workflows for e-Science: Scientific Workflows for Grids*. Springer Publishing Company, Incorporated.

Weyns, D. and Andersson, J. (2013). On the challenges of self-adaptation in systems of systems. In *1st SESoS*, SESoS '13, pages 47–51, New York, NY, USA. ACM.

Uma Arquitetura para Ecossistema de Software Científico

Vitor Freitas, José Maria N. David, Regina Braga, Fernanda Campos

Programa de Mestrado em Ciência da Computação (PGCC/DCC)
Universidade Federal de Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil
{vitor.freitas,jose.david, regina.braga, fernanda.campos}@ufjf.edu.br

Resumo. *A concepção de workflows científicos é uma abordagem utilizada no contexto de e-Science. Existem muitas pesquisas voltadas para o gerenciamento e execução de experimentos baseados em workflows. No entanto, experimentos complexos envolvem interações entre pesquisadores geograficamente distribuídos, demandando utilização de grandes volumes de dados, serviços e recursos computacionais distribuídos. Este cenário categoriza um ecossistema de experimentação científica. Para conduzir experimentos neste contexto, cientistas precisam de uma arquitetura flexível, extensível e escalável. Durante o processo de experimentação, informações valiosas podem ser perdidas e oportunidades de reutilização de recursos e serviços desperdiçadas, caso a arquitetura de ecossistema para e-Science não considere estes aspectos. Com o objetivo de tratar a extensibilidade de plataformas de ecossistemas, este trabalho apresenta uma arquitetura orientada a serviços apoiada por uma rede ponto a ponto, desenvolvida para tratar as etapas do ciclo de vida de um experimento científico. Este trabalho apresenta como contribuições uma arquitetura para ecossistemas de software científico, a implementação desta arquitetura, bem como a sua avaliação.*

Abstract. *The conception of scientific workflows is an approach used in the context of e-Science. There are many researches about the management and execution of experiments based on workflows. However, scientific experiments involve complex interactions between geographically distributed researchers, requiring the usage of large amount of data, services and distributed computing resources. This scenario categorizes a scientific experimentation ecosystem. In order to carry out experiments in this context, researchers need an architecture for e-Science that supports extensibility. During the experimentation process, valuable information can be unexploited and, as a result, reusing opportunities of resources and services could be lost if the ecosystem architecture for e-Science does not consider previous mentioned requirements. This works presents a service-oriented architecture supported by a peer-to-peer network. It was developed to support life-cycle stages of a scientific experiment. This work also presents, as contributions, an architecture to support experiments execution of scientific software ecosystems, as well as its evaluation.*

1. Introdução

Workflow científico é uma abordagem utilizada no contexto de e-Science e é relacionada a organização de um fluxo de execução de aplicações científicas, que devem ser sequenciadas de forma a realizar o experimento. (Altintas et al., 2004). A concepção de *workflows* científicos não é uma tarefa trivial, demandando um conhecimento especializado, muitas vezes interdisciplinar, exigindo do cientista algum conhecimento em computação. Como resultado, criam-se algumas barreiras como, a dificuldade no desenvolvimento e, sobretudo, na reutilização de workflows concebidos por outros cientistas, levando muitas vezes ao retrabalho.

O conceito de Linha de Produtos de Software (LPS) vem sendo utilizado neste contexto (Castro et al., 2015). A utilização de uma LPS no contexto científico pode auxiliar cientistas na concepção e controle de *workflows*. No entanto, o processo de experimentação vai além da concepção. Experimentos complexos envolvem interações entre pesquisadores, utilização de grandes volumes de dados, de serviços e de recursos computacionais distribuídos, constituindo um ecossistema de software científico.

Muitas vezes um experimento científico é uma atividade colaborativa. Ele passa por um ciclo de vida que se inicia na investigação do problema, seguindo pela prototipação e execução do experimento, até finalmente chegar à etapa de publicação das contribuições (Belloum et al., 2011). Durante o processo de experimentação, informações podem ser perdidas e oportunidades de reutilização de recursos e serviços desperdiçadas, caso a arquitetura de suporte para e-Science não considere estes aspectos.

Neste contexto, este trabalho define um Ecossistema de Software Científico (ECOSC), denominado ECOS-PL Science, como um subconjunto de Ecossistema de Software (ECOS), conforme definido por Jansen et al. (2009). Um ECOSC pode ser definido pelas suas relações com fornecedores de software científico, institutos de pesquisa, pesquisadores, órgãos de fomento, instituições financiadoras, e as partes interessadas nos resultados de pesquisa. Portanto, a arquitetura de um ECOSC deve ser flexível, uma vez que ela pode integrar com plataformas científicas externas, que evoluem de maneira independente, e estão em constante evolução. Estes relacionamentos ocorrem para gerar maior valor para o ECOSC, os quais requerem a abertura de suas fronteiras onde aplicações terceiras passam a se conectarem e se beneficiarem de seus serviços, gerando valor para as partes envolvidas. Portanto, a arquitetura do ECOSC precisa ser extensível. Um ECOSC além de ser provedor de serviços, é também um consumidor de serviços de software científico, sendo necessário que sua arquitetura esteja apta a realizar novas integrações sem que haja modificações substanciais na arquitetura da solução. Finalmente, a arquitetura de um ECOSC precisa ser escalável, uma vez que ela é extensível, podendo ocasionar em um crescimento repentino e inesperado de requisições pelos serviços.

A proposta deste trabalho é se ter um ambiente compartilhado, que possibilite: (i) a presença simultânea de cientistas trabalhando em um mesmo experimento, (ii) o tratamento de grandes volumes de dados relativos ao processo de experimentação, (iii) a execução de *workfows* científicos dentro da plataforma, e (iv) a viabilização da evolução da plataforma do contexto de e-Science. No contexto deste artigo, apenas o requisito não funcional extensibilidade foi avaliado.

Este trabalho está dividido da seguinte forma. A seção 2 apresenta o conceito de experimentação científica e os principais trabalhos relacionados. A seção 3 discute a abordagem ECOS PL-Science, e apresenta a arquitetura proposta como solução. A próxima seção (seção 4) descreve um estudo de caso e a aplicação de métricas para avaliação da extensibilidade da solução proposta. Finalmente, as considerações finais são apresentadas na seção 5.

2. Software Científico

Experimentos científicos complexos envolvem a utilização de dados e recursos computacionais distribuídos, demandando a colaboração de cientistas geograficamente distribuídos (Belloum et al., 2011). Com isso, uma nova geração de redes sociais de pesquisa surgem, como o myExperiment (Roure et al., 2009), proporcionando um ambiente colaborativo para descoberta, utilização e compartilhamento de *workflows* científicos. Belloum et al. (2011) descrevem o ciclo de vida de um experimento científico que se inicia na investigação do problema, seguindo pela prototipação e execução do experimento, até finalmente chegar à etapa de publicação das contribuições. Durante a etapa de investigação do problema é feita a busca por problemas relevantes, as ferramentas disponíveis são exploradas, objetivos são definidos e o problema é decomposto em etapas. Na etapa de prototipação do experimento o *workflow* é desenhado e os componentes necessários são desenvolvidos. Na etapa de execução do experimento ocorre a execução propriamente dita, controle, coleta e análise dos resultados. Finalmente os dados são anotados e as contribuições publicadas durante a etapa de publicação dos resultados.

Belloum et al. (2011) propõem um ciclo de vida do processo de experimentação científica colaborativa composto pelas etapas de investigação do problema, prototipação do experimento, execução e publicação dos resultados. Ainda que a colaboração e o compartilhamento de recursos ocorram, a abordagem não pode ser considerada um ECOSC, pois não atende aos requisitos de extensibilidade que uma plataforma de ECOSC demanda. Uma das ferramentas utilizadas pelo projeto com a finalidade de compartilhar arquivos faz uso de protocolos FTP (Grid-FTP e SSH-FTP), enquanto na proposta deste trabalho é implementada uma rede ponto a ponto, de modo a descentralizar o compartilhamento de arquivos. Outra diferença na proposta de Belloum et al. (2011) está na interoperabilidade dos recursos computacionais para apoiar as etapas do ciclo de vida de um experimento científico, e na ausência de um ambiente multiusuário.

A abordagem proposta por Mattoso et al. (2010) considera que um experimento científico passa por três etapas, tais como: composição, execução e análise. Ferramentas que dão suporte ao ciclo de vida de experimentação científico são discutidas, no entanto nenhuma integração ou ferramenta para apoiar todo processo é proposto. Zhang et al. (2014) propõem uma abordagem denominada Confucius, para o desenvolvimento de *workflows* científicos de maneira colaborativa, estendendo um gerenciador de *workflow* científico de código aberto. A abordagem tem o foco na atividade de composição de *workflows*, não tratando as etapas anteriores e posteriores do processo de experimentação. No mesmo sentido, a abordagem Co-Taverna é proposta por Zhang (2010), uma extensão do projeto Taverna. Novamente, a proposta não contempla todas as etapas de um experimento científico, embora contribua para o processo de experimentação colaborativo.

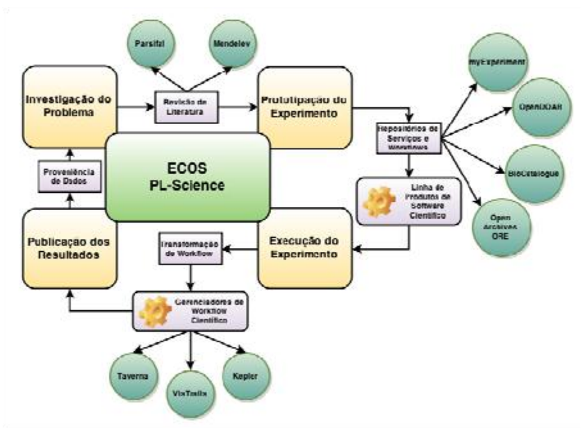


Figura 1. Ciclo de vida de um experimento científico no ECOS PL-Science

Mattoso et al. (2010) e Belloum et al. (2011) apresentam uma visão mais ampla do processo de experimentação, considerando as demais etapas de um experimento científico. Nenhum dos trabalhos relacionados utilizam uma abordagem de ECOS ou fazem uso de redes ponto a ponto, embora a proposta de Belloum et al. (2011) apresente alguns aspectos relacionados a ECOS.

3. ECOS PL-Science

Em sua essência, o ECOS PL-Science utiliza o conceito de ciclo de vida de um experimento científico proposto por Belloum et al. (2011). Explora cada etapa e provê recursos para auxiliar no processo de experimentação (Figura 1). O ciclo de vida foi adaptado, propondo a realização de revisões sistemáticas de literatura durante a etapa de investigação do problema e a utilização do conceito de uma LPSC na etapa de prototipação do experimento. Durante a etapa de prototipação são disponibilizados, aos cientistas, recursos para utilizarem *workflows* e serviços web de plataformas como myExperiment e BioCatalogue (Bhagat et al., 2010). A linha de produto de software científico Collaborative PL-Science é então incorporada na etapa de prototipação do experimento, aumentando o nível de reuso e a qualidade no desenvolvimento de *workflows* científicos, além de reduzir o tempo e consequentemente o custo do desenvolvimento. Finalmente, não somente os resultados da execução do experimento são armazenados no ECOS PL-Science, mas também todos os dados relativos ao processo de experimentação, possibilitando que outros pesquisadores possam consultar.

A arquitetura do ECOS PL-Science é apresentada na Figura 2. A camada de lógica de negócio é responsável pelo processamento, separando as responsabilidades pelo processamento dos dados da visualização, que neste ocorre via web. Com isto, a aplicação pode ser estendida para outras plataformas, como aplicações móveis por exemplo, sendo necessária somente a construção dos componentes de interface. A separação da visualização da lógica de negócio é importante para alcançar um nível maior de extensibilidade da plataforma, uma vez que a API se comunica com a camada de lógica de negócio para disponibilizar para aplicações externas, dados, funcionalidades e serviços da plataforma ECOS PL-Science.

A camada de visualização, ilustrada pela Figura 3 representa a aplicação web propriamente dita, onde os cientistas interagem com a aplicação em um ambiente multiusuário, executando atividades relativas a condução de um experimento científico.

A rede ponto a ponto trabalha diretamente no núcleo da aplicação, onde ocorre o gerenciamento dos artefatos da LPSC. Cada instância do ECOS PL-Science exerce o papel de um ponto na rede, compartilhando seus artefatos com outras aplicações, sendo estas conhecidas ou não. O núcleo da aplicação faz parte da proposta Collaborative PL-Science, onde elementos e serviços de colaboração foram associados à LPSC, onde estão representados os processos envolvidos na etapa de concepção de *workflows* científicos. A interação dos usuários externos ocorre em dois níveis, primeiro na camada de visualização da aplicação, atuando na condução de experimentos e no desenvolvimento de artefatos. O segundo nível ocorre no ambiente de desenvolvimento do ECOSC, gerenciado na plataforma GitHub. Através dela, desenvolvedores externos podem auxiliar na construção da plataforma, propondo melhorias e desenvolvendo novas funcionalidades. Essas funcionalidades serão avaliadas pela equipe de desenvolvimento interna da plataforma, podendo ou não serem integradas no código fonte. Cientistas ganham um canal de comunicação, através do qual podem solicitar novas funcionalidades ou reportarem problemas na plataforma.

Atualmente, o ECOS PL-Science está integrado com as plataformas Parsifal[1], Mendeley, Taverna Server (Zhang, 2010), myExperiment (Roure et al., 2009) e BioCatalogue (Bhagat et al., 2010). Por restrição de espaço, detalhes das integrações não são apresentados neste artigo[2]. No Com o desenvolvimento da proposta, conclui-se que são necessidades importantes para o sucesso de uma API: (i) a documentação completa de todos recursos disponíveis na API, (ii) a flexibilidade nos tipos de dados suportados, idealmente disponibilizar formatos XML e JSON, (iii) a coerência na implementação dos métodos HTTP (por exemplo, não utilizar método GET para recuperar dados e remover dados – neste caso método DELETE deveria ser utilizado), (iv) *sandbox*, ou um ambiente de desenvolvimento, para viabilizar os testes durante a integração, (v) a distribuição de software cliente de integração da API, (vi) estar engajado com a comunidade de desenvolvedores de código aberto, e (vii) dar suporte aos clientes de integração desenvolvidos por terceiros.

4. Avaliação da Solução Proposta

Um estudo de caso foi conduzido com o intuito de avaliar o requisito não funcional de extensibilidade da plataforma ECOS PL-Science. O escopo da avaliação foi definido com base no método GQM, descrito a seguir: **analisar** a arquitetura do ECOS PL-Science **com o propósito** de avaliar sua extensibilidade **sob o ponto de vista** dos desenvolvedores **no contexto** da evolução de um ECOSC. A partir do escopo, a questão de pesquisa foi definida: A arquitetura do ECOS PL-Science viabiliza a extensão de suas funcionalidades? A hipótese nula foi definida como: (**H0**) A arquitetura do ECOS PL-Science não viabiliza a extensão de suas funcionalidades. A hipótese alternativa foi definida como: (**H1**) A arquitetura do ECOS PL-Science viabiliza a extensão de suas funcionalidades. Com base na questão de pesquisa o estudo experimental foi planejado. A avaliação foi conduzida com um grupo de alunos de mestrado da UFJF cujos projetos de pesquisa estão diretamente ligados à evolução da plataforma ECOS PL-Science. Os participantes possuem conhecimento prévio da plataforma e começaram a atuar no

[1] http://parsif.al

[2] Maiores informações podem ser obtidas em http://pgcc.github.io/plscience.

desenvolvimento a partir da adoção de uma estratégia de ECOSC. O projeto tornou-se código aberto e é gerenciado pela plataforma GitHub.

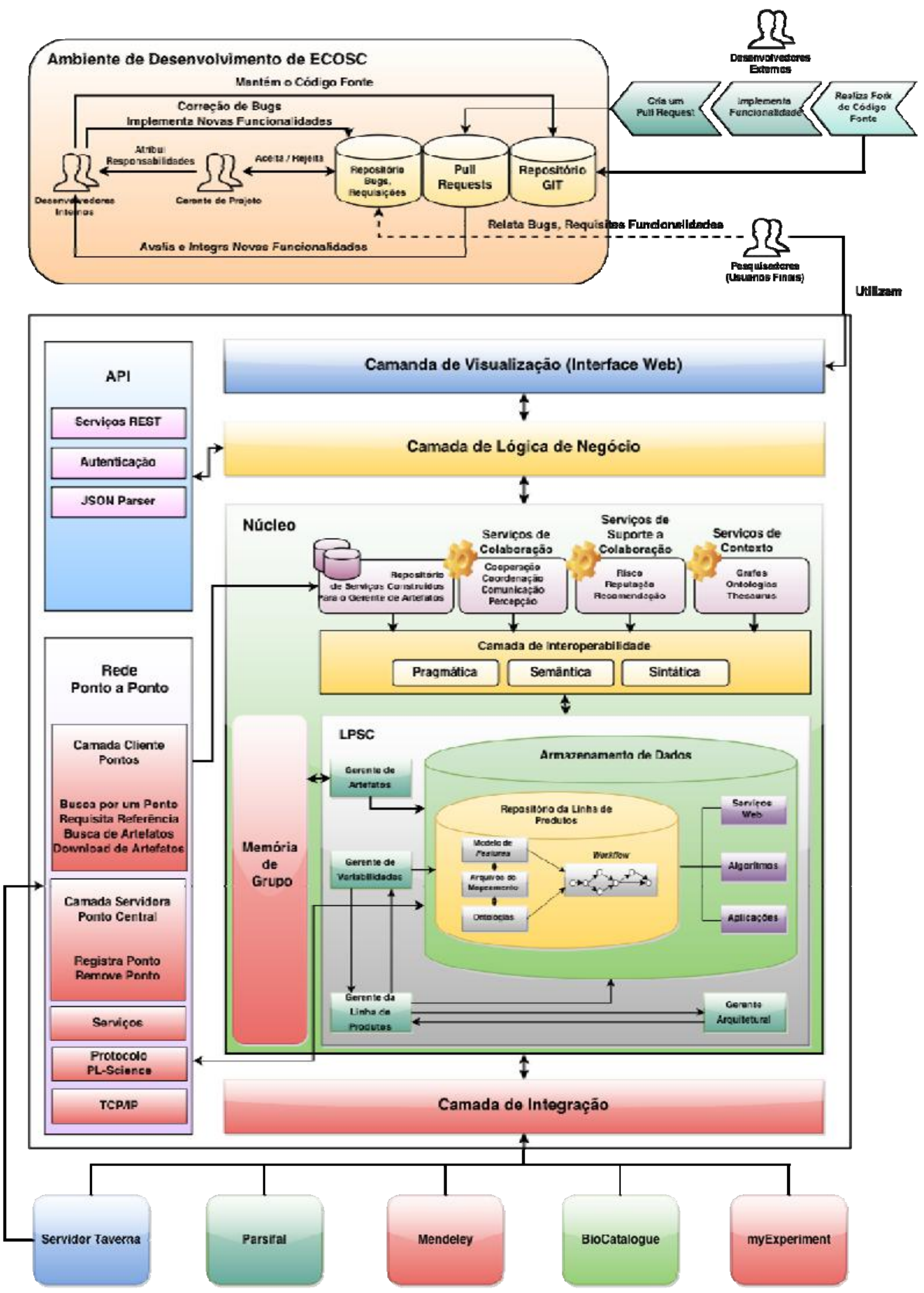
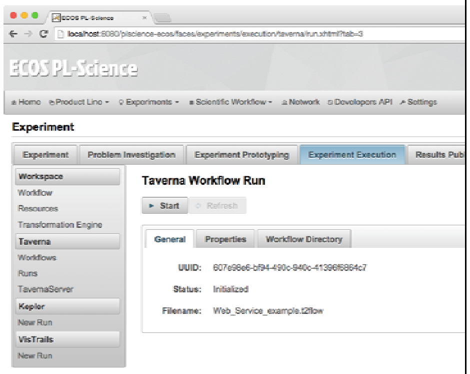
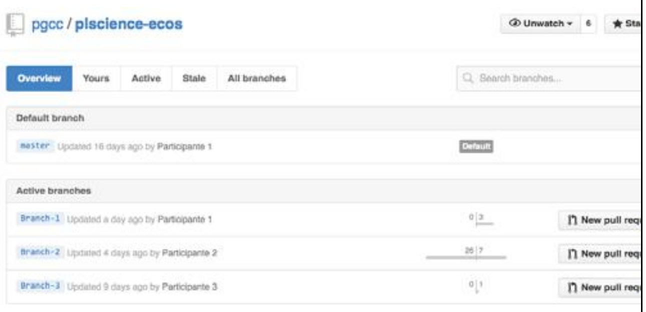


Figura 2. Arquitetura da Abordagem ECOS PL-Science



UUID: 607e98e6-bf94-490c-940c-41396f6864c7
Status: Initialized
Filename: Web_Service_example.t2flow

Figura 3. Interface Web ECOS PL-Science (visão parcial)



master Updated 16 days ago by Participante 1
Branch-1 Updated a day ago by Participante 1
Branch-2 Updated 4 days ago by Participante 2
Branch-3 Updated 9 days ago by Participante 3

Figura 4. *Branches* na plataforma GitHub

Na plataforma GitHub, cada participante criou um *branch*, conforme pode ser visto na Figura 4 (os nomes dos participantes foram omitidos), a partir da versão mais recente e as funcionalidades começaram a ser desenvolvidas: (i) inclusão de elementos de colaboração e comunicação durante todas as etapas do experimento; (ii) integração da plataforma com ontologias de colaboração; (iii) suporte à interoperabilidade pragmática no desenvolvimento colaborativo de *workflows*. Os participantes foram orientados a se limitarem à implementação de suas funcionalidades, e que nesse momento não realizassem refatorações de código, por exemplo.

As implementações ocorreram em momentos distintos, e todas elas representam trabalhos em andamento que estendem a abordagem ECOS PL-Science. Os participantes foram entrevistados e como fonte de coleta de dados adicional, dados históricos do GitHub foram avaliados. Um dos indícios de que a arquitetura é extensível, é a quantidade de código fonte existente que foi alterado, para que novas funcionalidades fossem implementadas. O GitHub proporciona uma visão com base nos *commits*, de quantas linhas de código foram incluídas e quantas foram removidas. Um alto número de linhas removidas sugere que muitas linhas de código precisaram ser alteradas e adaptadas para que a funcionalidade fosse implementada. Os resultados da extração dos dados são apresentados na **Erro! A origem da referência não foi encontrada.**1.

Tabela 1. Extração de dados do estudo de caso

Participantes	Num. de Commits	Arq. Alterados ou Incluídos	Adições	Remoções
Participante 1	3	17	1951	24
Participante 2	7	80	16280	1
Participante 3	1	25	806	40

Através da análise dos dados a questão de pesquisa pode ser respondida. Comparando o número de remoções de linhas de código com o número de adições, pode-se concluir que a arquitetura do ECOS PL-Science pode ser estendida sem que haja grandes alterações de sua estrutura. Como resultado, existem evidências de que a hipótese nula (H0) pode ser rejeitada e a hipótese alternativa (H1) pode ser aceita.

5. Conclusões

Pode-se destacar como contribuição deste trabalho, o desenvolvimento de uma arquitetura para ECOSC para auxiliar cientistas na condução de experimentos colaborativos. Através desta arquitetura, plataformas de softwares científicos são integradas em um ambiente multiusuário, de modo a satisfazer às necessidades do processo de experimentação. Além disso, oferece suporte durante as etapas de investigação do problema, prototipação do experimento, execução e a publicação dos resultados no ciclo de vida de um experimento. Para tanto, um ciclo de vida foi estendido com o objetivo de apoiar experimentos científicos.

Durante o processo de desenvolvimento da plataforma, bem como durante a elicitação dos requisitos, uma dificuldade foi encontrar especialistas no domínio, de modo a alinhar a solução com necessidades reais do domínio. Outra questão é que a rede ponto a ponto desenvolvida neste trabalho é um protótipo, desenvolvida somente para avaliar a viabilidade de sua adoção em um ECOSC.

Um estudo de caso foi conduzido com o intuito de avaliar o requisito não funcional de extensibilidade da plataforma ECOS PL-Science. Os resultados obtidos foram promissores mas não podem ser generalizados, sendo válidos para o contexto do ECOS PL-Science. Em relação à escalabilidade e flexibilidade seria necessário uma quantidade maior de experimentos e, portanto, não foram considerados neste experimento. Esses requisitos não funcionais devem ser tratados em trabalhos futuros.

Referências Bibliográficas

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S. (2004) “Kepler: an extensible system for design and execution of scientific workflows”, In: Scientific and Statistical Database Management. Proc. 16th Int.Conference on, p. 423–424.

Belloum, A. et al. (2011) “Collaborative e-Science Experiments and Scientific Workflows”, IEEE Computer Society vol.15, no. 4, pp. 39-47, doi:10.1109/MIC.2011.87.

Bhagat, J. et al. (2010) “BioCatalogue: A universal catalogue of web services for the life sciences”. Nucleic Acids Research, v. 38, p. 689–694.

Castro, G., Braga, R., David, J. M. N., Campos, F. (2015) “A Scientific Software Product Line for the Bioinformatics Domain”, JBI, v. 56, p. 239-264.

Jansen, S., Finkelstein, A., Brinkkemper, S. (2009) “A Sense of Community: A Research Agenda for Software Ecosystems”, ICSE’09, p.187-190.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V. (2010) “Towards supporting the life cycle of large scale scientific experiments”, IJBPIM, v. 5, p. 79-92,.

Roure, D., Goble, C., Stevens, R. (2009) “The Design and Realization of the myExperiment Virtual Research Environment for Social Sharing of Workflow”, Future Generation Computer Systems, p. 561–567.

Zhang, J. (2010) “Co-Taverna: A tool supporting collaborative scientific workflows”, IEEE 7th International Conference on Services Computing, SCC 2010, p. 41–48.

Zhang, J., Kuc, D., Lu, S. (2014) “Confucius: A tool supporting collaborative scientific workflow composition”, IEEE Trans. on Services Computing, 7(1), p. 2–17.

Ecossistema de Software no Contexto do Poder Judiciário - Apontamentos Sobre o ECOS Projudi no Estado do Paraná

Rebeca Teodoro da Silva[1, 2], Luiz Gustavo Ferreira Aguiar[1,2], Elias Canhadas Genvigir[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR) – Cornélio Procópio – PR

[2]Tribunal de Justiça do Paraná (TJ PR)

{rebeca.teodoro, gustavo.bytes}@gmail.com, elias@utfpr.edu.br

Abstract. *Software Ecosystem (ECOS) is composed of business systems and subsystems that interact in a specific market niche. This article presents an initial discussion of the ECOS in a real scenario the Brazilian Judiciary in Projudi System, which is currently used almost all Brazilian states. The ECOS Projudi has a complex web of interconnections between systems, actors, technical, transactional and social. It held an initial discussion about a specific scenario of Paraná Judiciary presenting the technical, transactional and social Projudi the system. This discussion is aimed at exploring the new research possibilities about this type of ECOS.*

Resumo. *Ecossistema de Software (ECOS) é composto por sistemas e subsistemas de negócio que se interagem em um nicho específico do mercado. Este artigo apresenta uma discussão inicial sobre o ECOS em um cenário real do Poder Judiciário Brasileiro no Sistema Projudi, que é atualmente utilizado pela maioria dos estados brasileiros. O ECOS Projudi possui uma complexa rede de interligações entre sistemas, atores, elementos técnicos, transacionais e sociais. Realiza-se neste trabalho uma discussão inicial acerca de um cenário específico do Poder Judiciário do Paraná apresentando os elementos técnicos, transacionais e sociais do Sistema Projudi. Tal discussão tem por meta apresentar as possibilidades de pesquisa a serem exploradas neste tipo de ECOS.*

1. Introdução

Um ecossistema de software (ECOS) pode ser definido como um conjunto de atores funcionando como uma unidade que interage com um mercado distribuído entre software e serviços, juntamente com as relações entre eles, frequentemente apoiadas por uma plataforma tecnológica ou por um mercado comum, operando através da troca de informações, recursos e artefatos (Jansen et al., 2009). Em outras palavras, um ECOS é uma interação de um conjunto de atores sobre uma plataforma tecnológica comum, que resulta em um número de soluções ou serviços de software (Manikas & Hansen, 2013).

Os conceitos de ecossistemas de software estão em torno de praticamente todas as soluções de software de sucesso (Jansen et al., 2009). Esses ecossistemas consistem basicamente de elementos como um centralizador, uma plataforma que pode ser uma tecnologia ou o mercado e os agentes do nicho relacionado (Santos et al., 2013).

O desenvolvimento de tais sistemas envolve, mesmo sem o conhecimento prévio dos envolvidos, elementos técnicos, transacionais e sociais, que são considerados como dimensões do desenvolvimento de ECOS (Santos e Werner (2011,2012)). Neste contexto, este trabalho tem como objetivo analisar como o sistema de apoio as atividades do Poder Judiciário no Paraná e de outros 18 estados brasileiros, denominado sistema Projudi, pode ser considerado como um Ecossistema de Software. Tal cenário permite a composição de sistemas que estão interligados, que são influenciados por atores internos e externos e que apresentam relações de transferência de transações e contextos sociais, ou seja, elementos técnicos, transacionais e sociais.

Assim, este trabalho apresenta uma discussão inicial sobre ecossistema de software no contexto jurídico, visando identificar as interligações entre os diversos elementos e a natureza dos ECOS. O artigo está organizado da seguinte forma: a seção 2 apresenta o contexto do Poder Judiciário, o Processo Judicial Digital e os elementos transacionais, técnicos e sociais deste contexto na ferramenta Projudi; a seção 3 apresenta uma análise do ECOS Projudi no Poder Judiciário; e a seção 4 as considerações finais.

2. O Contexto do Poder Judiciário e o Processo Judicial Digital

A organização do Sistema Judiciário Brasileiro é baseada numa combinação de natureza judicial, localização física e grau de jurisdição. Tais características permitem que diversas combinações de contexto sejam criadas, com opções que vão além do número de unidades administrativas autônomas. Visto que o Brasil é uma república federativa, o Sistema Judiciário também se encontra organizado em nível estadual (Andrade & Joia, 2012), tal como apresentando na Figura 1.

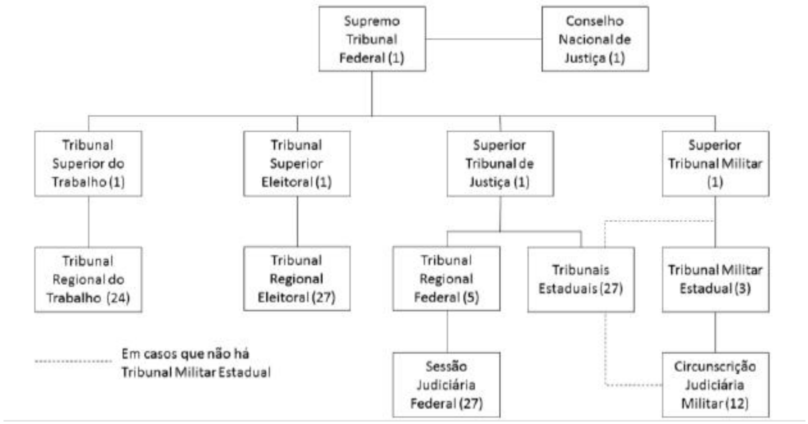

Supremo Tribunal Federal (1)
Conselho Nacional de Justiça (1)
Tribunal Superior do Trabalho (1)
Tribunal Superior Eleitoral (1)
Superior Tribunal de Justiça (1)
Superior Tribunal Militar (1)
Tribunal Regional do Trabalho (24)
Tribunal Regional Eleitoral (27)
Tribunal Regional Federal (5)
Tribunais Estaduais (27)
Tribunal Militar Estadual (3)
Em casos que não há Tribunal Militar Estadual
Sessão Judiciária Federal (27)
Circunscrição Judiciária Militar (12)

Figura 1 - Estrutura Organizacional do Poder Judiciário. Fonte: Adaptado de (Andrade & Joia, 2012)

A informatização dos sistemas da área jurídica passa pelo conceito do processo judicial digital, também chamado de processo virtual ou de processo eletrônico. Tal elemento tem como premissa, gerenciar e controlar os trâmites de processos judiciais nos Tribunais de forma eletrônica, reduzindo tempo e custos. O principal intuito é a completa informatização da justiça, retirando burocracia dos atos processuais, o acesso imediato aos processos, bem como a melhoria no desempenho das funções próprias de cada usuário, o mesmo acessa somente o módulo que ofereça as funções que ele necessita para desenvolver suas atividades (CNJ, 2015).

O Projudi foi constituído como uma iniciativa denominada Processo Digital no Juizado do Consumidor (Prodigicon) e era um projeto de conclusão de curso de dois estudantes de Ciências da Computação da Universidade Federal de Campina Grande, cuja abrangência de aplicação era restrita a processos de juizados especiais cíveis em matéria de direito do consumidor (Andrade, 2013). Em 2005, teve seu nome alterado para Projudi e foi instalado em outras unidades do Tribunal de Justiça do Estado da Paraíba. No ano de 2006 seus autores fizeram a doação ao Conselho Nacional de Justiça - CNJ do código base do sistema (CNJ, 2006).

Por sua vez, o CNJ estabeleceu as premissas para a criação do Processo Judicial Digital e mantém, atualmente, o código fonte do Sistema Projudi que pode ser caracterizado como um software de tramitação de processos judiciais. Seu uso encontra-se em franca expansão nos os estados do Brasil, sendo que, atualmente, 19 dos 27 estados brasileiros fazem uso deste sistema. Aponta-se também que o CNJ, conjuntamente com seus técnicos, disponibilizou aos Tribunais Estaduais este sistema, que se caracteriza como um sistema de código aberto para uso em ambiente Web desenvolvido em linguagem Java. O nome “PROJUDI” decorre das iniciais de Processo Judicial Digital (CNJ, 2015). Porém Andrade (2013) observa que, ao difundir o Projudi entre os tribunais, o CNJ optou por fazê-lo sem seu desenvolvimento estar completo distribuindo o código-fonte e a documentação, para que os próprios tribunais pudessem promover o aprimoramento do sistema e adaptação às realidades locais, em especial a conexão com base de dados locais previamente existentes. Essa abordagem permitiu a difusão de diferentes arranjos de desenvolvimento e adaptação do sistema.

Os estudos da área do direito ou da administração judiciária abordam o Projudi como um fenômeno único, com o foco de interesse dos pesquisadores voltado à relação entre o sistema e o próprio processo judicial, com eventuais ganhos de produtividade para o processo ou a Justiça (Andrade, 2013). Contudo a informatização do processo judicial envolve a compreensão de uma complexa rede de elementos transacionais, técnicos e sociais. Tais elementos, que norteiam o desenvolvimento de ECOS no contexto jurídico, estão resumidos e apresentados na Figura 2.

Os elementos transacionais observados no ECOS Projudi tratam sobre a eficiência, por meio do gerenciamento e controle dos trâmites de processos judiciais e da produtividade devido a automatização dos atos jurídicos reduzindo tempo e custos.

Os elementos técnicos visam facilitar o envolvimento de diversos atores que são internos e externos. Pode-se citar como atores internos o Processo Judicial (atos jurídicos) e o Software (modelagem, desenvolvimento e manutenção); e como ator externo a Administração Judiciária (leis, portarias e análises técnicas e normativas). Em relação aos atores internos, pode-se apontar também a participação de diversos envolvidos na área jurídica, tais como: advogados, juízes, promotores, conselhos, desenvolvedores de software, servidores, tribunais e até mesmo a sociedade de maneira geral. Já os atores externos, como por exemplo as leis, influenciam e delimitam o nicho de negócio da área jurídica e por consequência contribuem com ECOS Projudi.

O principal elemento social envolvido no ECOS Projudi envolve o acesso, ou seja, objetiva facilitar o acesso e as interações de pessoas que são vinculadas a área do direito, mas que não necessariamente possuem um interesse em comum e em seguida o conceito de transparência dos atos públicos que se dá pela normatização/padronização dos tramites de processos judiciais.

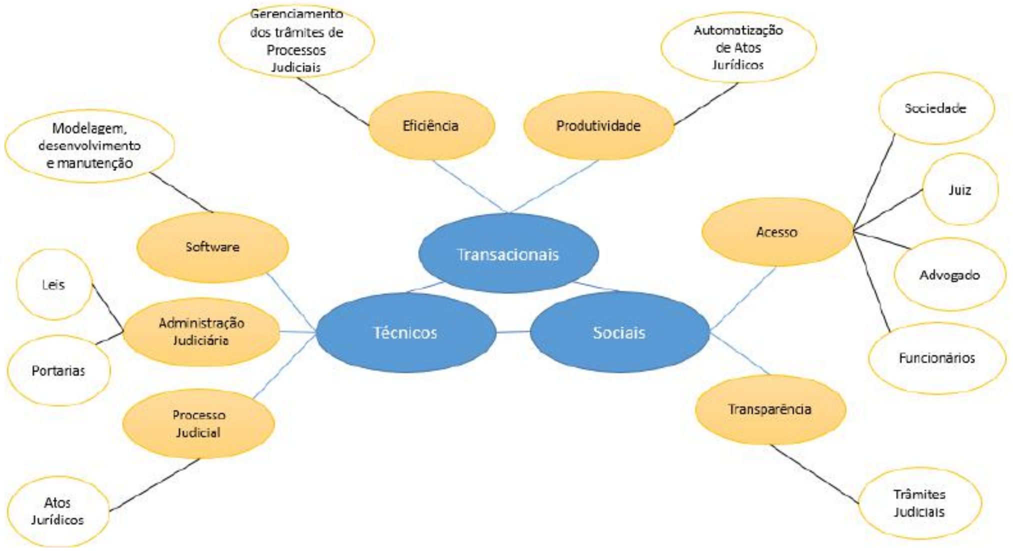


Figura 2 - Rede de elementos da informatização do processo judicial

3. Análise do ECOS no Poder Judiciário

A iniciativa de desenvolvimento de sistemas para o gerenciamento dos trâmites jurídicos advém tanto das questões normativas estabelecidas pelo estado quanto das ações individuais dos profissionais das áreas do direito e de desenvolvimento de sistemas. Especificamente o Projudi é um sistema que pode ser utilizado através da Internet e permite a completa substituição do papel por autos processuais digitais. Todo o documento enviado recebe um protocolo eletrônico e uma assinatura digital, certificando a origem e garantindo o conteúdo. Tal sistema possui como objetivo agilizar a Justiça; diminuir custos; aumentar a capacidade de processamento de ações; facilitar o trabalho dos advogados; melhorar a qualidade do atendimento às partes (Projudi, 2015).

Sobre as formas de acesso o Projudi permite que usuários cadastrados tenham acesso ao sistema. A consulta e a prática de atos processuais podem ser realizadas na Internet ou na sede do Juizado Especial. Os advogados que se cadastrarem recebem senha de acesso ao sistema e também certificados digitais que promovem a: identificação, segurança, autenticidade e fidedignidade dos documentos (Projudi, 2015).

Em relação às vantagens oferecidas pelo Projudi pode-se apontar a redução de barreiras/fronteiras para acesso aos trâmites; a disponibilidade o acesso instantâneo e remoto aos dados dos processos de qualquer lugar do mundo, via Internet; redução de custos visto que advogados podem acessar remotamente os processos podendo praticar atos processuais além da redução de despesas na administração dos processos e; agilidade na tramitação dos processos como em casos em que Juízes podem resolver questões urgentes mesmo sem comparecer à sede da Justiça (Projudi, 2015).

Em relação às interações o Projudi possui várias interações, visando fins específicos, entre o sistema e os perfis de atores envolvidos. Todavia, destacam-se três perfis genéricos: advogado, juiz e servidor de secretaria. Assim, o advogado pode ter a atribuição de cadastrar a ação no sistema Projudi, receber intimações e peticionar no processo. O juiz pode despachar no processo, ordenar e aprovar atos. O servidor de secretaria pode ter atribuições relacionadas à parte administrativa da ação, por exemplo, o servidor no sistema Projudi pode verificar as guias pagas relacionadas ao processo, enviar uma petição para o juiz apreciar e intimar o advogado.

Além disso, alguns sistemas são interligados ao sistema Projudi no Tribunal de Justiça do Paraná - TJPR. Por exemplo, no TJPR é utilizado um sistema chamado Sistema Uniformizado para a normatização de recolhimento de custas e despesas processuais que pode ser consultado pelo servidor de secretaria para a comprovação de recolhimento de custas de diversos atos. Assim, os comprovantes gerados pelo Sistema Uniformizado devem ser incluídos no Sistema Projudi.

No TJPR existem também sistemas que são voltados para a penhora de bens, tais como: o Bacenjud e o Renajud. O Bacenjud é um sistema eletrônico de comunicação entre o Poder Judiciário e as instituições financeiras e o sistema Renajud interliga o Poder Judiciário e o Departamento Nacional de Trânsito – Denatran (Bacenjud, Renajud e Infojud, 2015). Os comprovantes de efetivação da ordem judicial proferida no Sistema Projudi em ambos os sistemas, Bacenjud e Renajud, devem ser anexados no Sistema Projudi.

Outra interligação de sistema existente no sistema Projudi são os de busca de informações referentes a Receita Federal do Brasil – Sistema Infojud - e o das informações oriundas no Cadastro Nacional de Eleitores – Sistema SIEL. Quaisquer buscas realizadas nestes sistemas devem ter o resultado anexado no sistema Projudi ou disponibilizado para consulta pela parte. Embora sejam completamente distintos, estes sistemas fornecem informações relevantes para o processo eletrônico digital.

Esta interligação de sistemas pode ser analisada como características de um ecossistema de software, tal como apresentado na Figura 3. O conjunto de sistemas autônomos que integram o contexto jurídico é abrangente e podem possuir objetivos únicos e específicos.

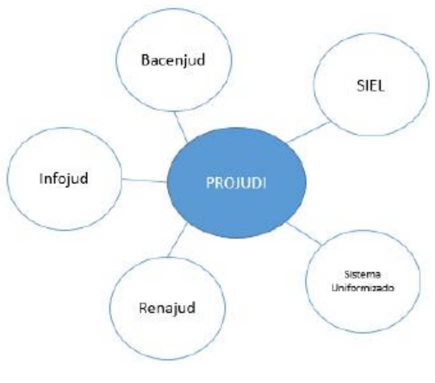


Figura 3 - Interligação entre sistemas com o Sistema Projudi

Na figura 4 é apresentado um cenário de interligação entre estes sistemas com o Sistema Projudi. O cenário é exemplificativo e se deve ao fato dos atos jurídicos não estarem representados em sua completude, embora estes atos estejam apresentados na figura eles são mostrados de uma forma simplista por existirem outros papéis que não são abordados neste estudo, como por exemplo o assessor jurídico, técnico judiciário, analista judiciário e o estagiário. Além disso, no Poder Judiciário existem vários tipos de processo judiciário, com leis específicas e que possuem características que não são inerentes a todos os processos. Assim, optou-se por apresentar um cenário exemplificativo, observado pelos autores, apenas para efeitos de demonstração das interligações que existem, mas que não necessariamente ocorre em todos os processos na ordem apresentada.

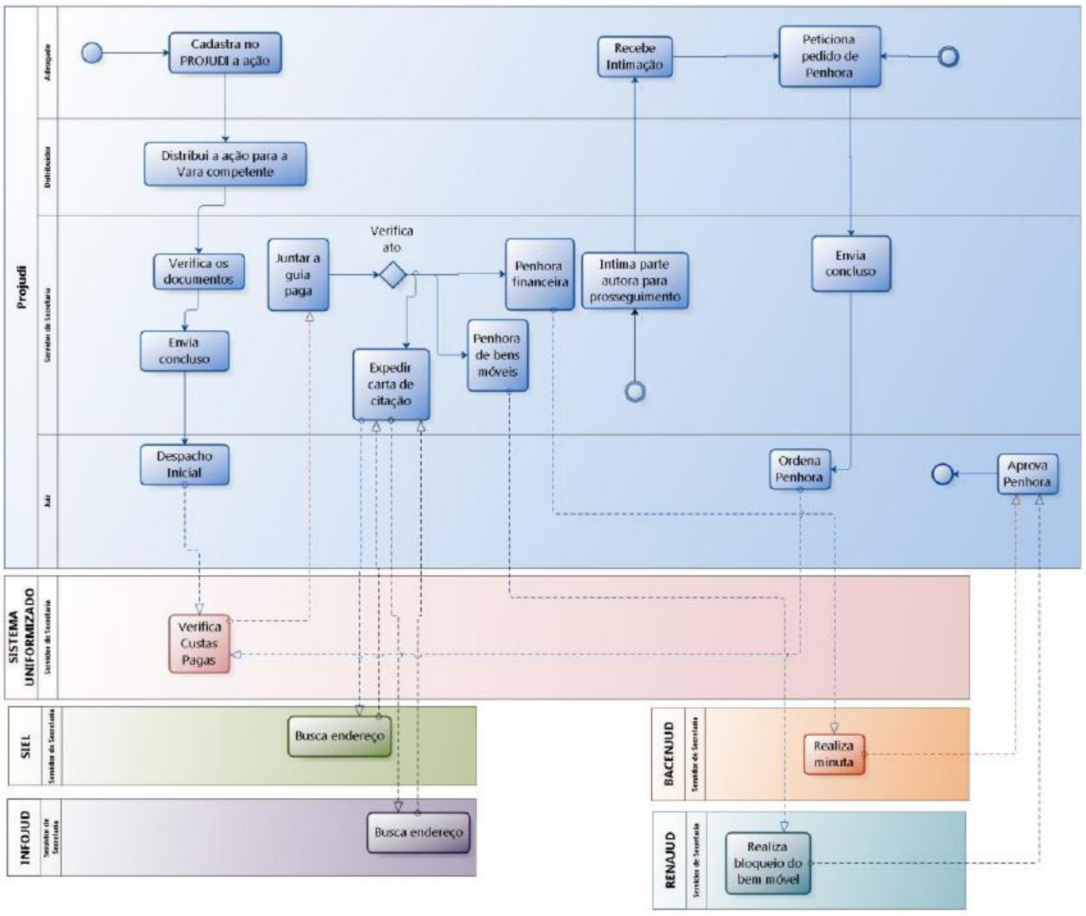


Figura 4- Cenário exemplificativo de interação dos sistemas

3.1 Considerações sobre o Ciclo de Vida

Sobre o ciclo de vida observa-se na literatura (Santos et al., 2013) duas linhas principais sobre as fases que compõem o desenvolvimento de um ECOS. Essas duas linhas apontam para dois contextos, com base no quesito transação, que as diferenciam. O primeiro caso é analisado sobre a premissa que o tipo de transação existente segue características comerciais (Jansen et al., 2009b). Tal modelo é dividido em quatro fases sendo elas: (1) o estabelecimento de um relacionamento de mercado com uma empresa dominante; (2) o surgimento de uma rede preliminar; (3) a diminuição do poder da empresa dominante e o estímulo das comunidades e, (4) a manutenção de uma comunidade de criação do ECOS. O segundo caso tem como premissa que se a análise da transação considera que esta não é estritamente comercial, como é o caso do sistema Projudi. Tal ciclo de vida pode ser analisado em quatro fases - o aprofundamento deste estudo é apresentado por Santos et al. (2013) estendido de Santos et al. (2012). Nesse último caso têm-se as seguintes fases: 1) iniciação – envolve a criação de elementos iniciais de caráter social; 2) propagação - é caracterizada pela adesão de novos atores e artefatos e a diminuição do poder do orquestrador; 3) automação – onde há um certo grau de coordenação entre as atividades de diferentes unidades administrativas; 4) amplificação – apresentação de uma estrutura auto–organizável e manutenção de uma comunidade calcada na rede de atores e de artefatos de um dado ECOS, onde não existe nenhum orquestrador dominante e o poder é distribuído; e por fim 4) terminação – encerramento do serviço ou substituição deste por outro.

Ao considerar o processo de automatização do Poder Judiciário Brasileiro observa-se que este pode ser dividido em 3 etapas organizacionais distintas: i) pré-automação, marcada por iniciativas individuais; ii) automação, onde há um certo grau de coordenação entre as atividades de diferentes unidades administrativas, permitindo algumas rotinas básicas como distribuição de petições iniciais, rotinas de publicação e acompanhamento processual; e iii) a terceira e atual fase, a virtualização de processos judiciais, ou simplesmente, sistemas de "processo eletrônico" (Andrade & Joia, 2010).

Ao analisar o ciclo de vida do ECOS Projudi observa-se, assim como outros sistemas que envolvem instituições públicas e, que em muitos casos, envolvem também relações com entidades privadas e com a sociedade, a presença de características mistas entre as fases dos dois ciclos de vida definidos pelo contexto do tipo de transação apresentados. Ou seja, o ciclo de vida de ECOS, no contexto do serviço público, independente do poder envolvido, pode apresentar fases que existam em ambos os ciclos que consideram análise do tipo de transação como norteador, ou seja, considerando a classificação apenas como ciclo com transações comerciais ou não comerciais. Como por exemplo, a fase de Rede Preliminar e Redução de Centralização apresenta-se, referente a ciclos comerciais está presente no caso do Projudi enquanto as fases de Iniciação, Propagação e Automação também estão.

4. Considerações Finais

Este artigo aborda uma discussão inicial sobre considerações do ECOS Projudi no Poder Judiciário. Neste contexto, existem diversos sistemas e atores que estão interligados e que sofrem a ação de atores externos e elementos técnicos, transacionais e sociais que formam uma complexa rede de interações.

Também foi apresentado um cenário exemplificativo para análise na competência cível do Poder Judiciário do Paraná considerando o relacionamento entre sistemas que podem constituir nichos específicos. Além disso, ainda são necessárias pesquisas que se aprofundam neste tema apresentando um maior aprofundamento sobre a implantação e a evolução dos canais próprios de troca de informação dentro da área Jurídica e como tais elementos afetam esse tipo de ECOS.

Pesquisas anteriores apontam que o Processo Judicial Eletrônico pode ser analisado como um ecossistema (*assemblages*) no contexto da área de administração (Martinez, 2012). Tais pesquisas instigam a continuidade dos trabalhos no contexto dos ECOS e como próximos passos, pretende-se investigar tais características.

6. Referências Bibliográficas

Andrade, A. S. C. G. (2013) "Trajetórias de Implantação do Projudi à Luz da Teoria Ator-Rede". 361f. Tese (Doutorado em Administração). Escola Brasileira de Administração Pública e de Empresas da Fundação Getúlio Vargas, Rio de Janeiro.

Andrade, A.; Joia, L. A. (2010) “Organizational Structure and ICT Strategies in the Brazilian Justice”. Proceedings of the 4th International Conference on Theory and Practice of Electronic Governance. ICEGOV. New York, NY, USA: ACM. Disponível em: <http://doi.acm.org/10.1145/1930321.1930345>

Andrade, A.; Joia, l. A. (2012) “Organizational structure and ICT strategies in the Brazilian Judiciary System”. Government Information Quarterly, v. 29, Supple, n. 0, pp. 32-42.

Bacenjud, Renajud e Infojud. (2015) “Bacenjud, Renajud e Infojud”. Disponível em: <http://www.tjpa.jus.br/PortalExterno/institucional/Corregedoria-do-Interior/76-BACENJUD---RENAJUD-E-INFOJUD.xhtml>. Acessado em: 03 de jun. 2015.

Brasil. (2006) “Lei No 11.419, de 19 de Dezembro de 2006”. Disponível em: <http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2006/Lei/L11419.htm>

Conselho Nacional De Justiça – CNJ. (2015) “Sistema CNJ-Projudi”. Disponível em: <http://www.cnj.jus.br/sistemas/projudi>. Acessado em: 18 de jun. 2015.

Conselho Nacional De Justiça – CNJ. (2015) “Termo de Doação de Software”. Brasília, 2006. Disponível em: < http://www.cnj.jus.br/images/stories/docs_cnj/termo_coop/doacao_software.pdf>. Acessado em: 15 jun. 2015.

Jansen, S., Brinkkemper, S., Finkelstein, A. & Bosch, J. (2009) “Introduction to the Proceedings of the First Workshop on Software Ecosystems”. In Proceedings of the First Workshop on Software Ecosystems (pp. 1-2), CEUR-WS.

Jansen, S.; Finkelstein, A.; Brinkkemper, S. (2009) “A sense of community: A research agenda for software ecosystems”. 2009 31st International Conference on Software Engineering - Companion Volume, pp. 187–190.

Manikas, k.; Hansen, K. M. (2013) “Software ecosystems – A systematic literature review”. Journal of Systems and Software, v. 86, n. 5, pp. 1294–1306.

Martinez, R. H. (2012) “Processo judicial eletrônico: uma abordagem metodológica para o processo de sua implementação”. Faculdade de Economia, Administração e Contabilidade, USP, São Paulo.2012.

Projudi. (2015) “Processo Judicial Digital”. Disponível em: < https://projudi.tjpr.jus.br/projudi/informacoesExtras/explicaProcessoDigital.htm>. Acessado em: 03 de jun. 2015.

Santos, R. P. Werner, C., M., L. (2011) “Treating Business Dimension in Software Ecosystems”. In: Third ACM/IFIP International Conference on Management of Emergent Digital Ecosystems, San Francisco, USA. pp. 197-201.

Santos, R. P. Werner, C., M., L. (2012) “Treating Social Dimension in Software Ecosystems through ReuseECOS Approach”. In: Sixth IEEE International Conference on Digital Ecosystem Technologies, Campione d’Italia. pp. 1-6.

Santos, R. P., Werner, C. M. L., Alves, C. F., Pinto, M. J. S., Cukierman, H. L., Oliveira, F. M. A., Egler, T. T. C. (2013) “Ecossistemas de Software: Um Novo Espaço para a Construção de Redes e Territórios Envolvendo Governo, Sociedade e a Web”. In: Werner, C.M.L.; Oliveira, F.J.G.; Ribeiro, P.T.. (Org.). “Políticas Públicas: Interações e Urbanidades”. 1ed.Rio de Janeiro: Letra Capital, pp. 337-366.

An Analysis of Dynamic Strategies during the Lifecycle of Software Ecosystems: The DS-SECO Model

Rodolfo V. C. L de Andrade[1], Carina Frota Alves[2], George Valença[2,3]

[1]FATECS – Centro Universitário de Brasília (Uniceub)
70.790-075 – Brasília – DF – Brazil

[2]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50.740-560 – Recife – PE - Brazil.

[3]Departamento de Informática
Universidade Federal Rural de Pernambuco (UFRPE) – Recife, PE – Brazil

`rodolfo.andrade@uniceub.br, {cfa,gavs}@cin.ufpe.br, georgevalenca@deinfo.ufrpe.br`

Abstract. *Software ecosystem (SECO) represents a trend in the software industry, which covers technical, social and business aspects of software development. In a SECO, companies must cooperate and compete to thrive. In this context, establishing an effective business strategy is an essential goal during the lifecycle of a SECO. This paper proposes the DS-SECO (Dynamic Strategies for Software Ecosystems) model, which is based on principles from strategy dynamics and the adoption of biological analogies. This model investigates how strategies can be adopted to increase SECO sustainability and generate performance enhancements that will keep the ecosystem healthy. We illustrate the use of DS-SECO model with an analysis of the iPhone SECO.*

1. Introduction

Software is a highly pervasive sector that influences the characteristics of products, processes and services in almost every industry [Yier et al. 2006]. In the software industry, most organizations do not have all resources needed to satisfy their customers. Organizations must now engage in a new perspective considering both themselves and third parties.

Inspired by properties of Natural ecosystems, researchers have coined a new term to analyse the software industry: Software Ecosystem (SECO) [Messerschmitt and Szyperski 2003]. The advent of SECOs influenced major players in the software industry to rethink their operating practices by opening their platforms to external players to attain business goals [Campbell and Ahmed 2010]. When joining a SECO, companies benefit from cost reduction, risk sharing and higher customer satisfaction [Jansen et al. 2009b]. Nevertheless, being part of a SECO also involves risks and challenges. In this context, establishing effective business strategies is an important goal during the SECO lifecycle.

This paper proposes the DS-SECO model, which considers principles from Strategy Dynamics [Warren 2002] to investigate how strategies can be adopted to increase SECO sustainability. The model provides a dynamic analysis of challenges and opportunities to define and assess business strategies during SECOs lifecycle. An analysis of the iPhone ecosystem is presented to illustrate how the model can be used.

2. Literature review

2.1. Key concepts

Component-based software development became commonplace. Software vendors develop products by integrating components developed by other companies [Santos and Werner 2010]. Moreover, companies can delegate several software development activities to partners [Jansen et al. 2007].

According to [Bosch 2009], a SECO is a set of software solutions that meet customer needs, without putting aside the relationships that exist among the suppliers and the customers. By joining all these viewpoints, it is possible to conclude that a SECO encompasses a strong interaction of several players with common objectives.

Three main roles can be highlighted in a SECO based on the classification proposed by [Iansiti and Levien 2004]. Firstly, **keystones** are companies that act as enablers and stabilizers of the ecosystem. Secondly, **niche players** are the majority of players in the ecosystem and make use of resources provided by the keystone. Finally, **dominators** attempt to control the ecosystem by making use of other companies' resources without a reciprocal benefit.

2.2. Software Ecosystems Lifecycle

Based on evolutionary stages proposed by [Moore 1993], a SECO lifecycle has four stages. The **Birth** stage focuses on the definition of customer needs and involves the initial development of products and services, with potential players joining the ecosystem to participate. The **Expansion** stage involves internal and external battles, the reach of new markets and market share segmentation. The **Leadership** stage happens after the ecosystem proves to be profitable. During this stage, internal disputes may occur among participants to get more power in the ecosystem. Finally, the **Self-renewal** stage means the ecosystem will either die or start a new evolutionary cycle by adopting novel technologies or adapting its business model.

As shown in [Hartigh et al. 2013], the health of an ecosystem is a way of assessing its strength at a specific moment. To measure it, three elements should be considered. **Productivity** indicates ecosystem ability to transform inputs into products and services. **Robustness** indicates the ecosystem capacity to deal with interferences and competition pressure. Finally, **niche creation** represents the possibilities to create opportunities for ecosystem participants.

2.3. Strategy Dynamics

According to [Warren 2002], decision-making needs a fact-based analysis to help increasing business performance. In this sense, strategies are seen as a set of decisions and actions to reach organizational goals that can affect organizational performance.

Strategy Dynamics provides understanding on strategic performance evolution. The approach emphasises building and sustaining resources as well as capabilities for companies to succeed. This is materialised in the Dynamic Resource System View of Strategy (**DRSV**) framework [Warren 1999]. The DRSV is based on the principle that performance is highly influenced by resources, which can be accumulated, consumed, depleted or decayed. The Mystrategy tool [Strategy Dynamics Website 2012] was designed to support DRSV for modelling strategy and performance.

Strategy dynamics proposes two artefacts to examine company performance: resources map and time-path graphs. Based on these artefacts, strategy analysts are able to answer three key strategic questions: **Why** has the historical business performance followed the time-path that it has? **Where** will the path of future performance take us in case we keep up as we are? **How** can we alter that future for the better?

3. The Dynamic Strategies for Software Ecosystems (DS-SECO) Model

As ecosystems are dynamic and change throughout time, the DS-SECO model was developed to enable an evolutionary analysis of the SECO to tackle strategic decisions along its lifetime. The model adopts a four-stage lifecycle and assesses SECO strategic performance in terms of health elements, as described in Section 2.2. To enable DS-SECO application in a feasible manner, the model was divided in five phases. They were adapted from DRSV [Warren 1999][Warren 2002] and should be carried out at the end of each lifecycle stage.

3.1. DS-SECO phase 1: Time-path definition

Initially, it is necessary to establish a time-path graph to analyse performance across time. The DS-SECO uses the lifecycle stages as time scale and SECO health elements as measures. The outcome of Phase 1 consists of three time-path graphs, one for each health element. The analysis of each ecosystem must define appropriate metrics to measure each health element, for instance [Hartigh et al. 2013][Jansen 2014] present a set of health metrics. Each graph has both a desired and feared situation that can change every time phase 1 is executed. Additionally, an analysis point separates past from future analysis. This point should be set at the end of each lifecycle stage to enable full assessment of alternative strategies. It is worth noting that the graphs do not need to have precise measurements as [Warren 2002] states that precise numbers are frequently unknown. In Section 4.1 (Figure 1), a time-path graph is described for the iPhone study.

3.2. DS-SECO phase 2: Resources identification

Key resources must be identified to create the resources list. They are inputs for the productive process and their management is essential for competitive advantage [Hartigh 2006]. In our proposal, resources belong to the SECO as a whole and are not associated with specific firms (as established originally in DRSV). By adopting this view, we focus on the strategies that increase ecosystem health instead of looking at specific strategies for individual companies. Ecosystem resources can come from the keystone, niche players or alliances among SECO participants. They can be tangible (e.g. capital, number of applications) or intangible (e.g. keystone reputation, staff expertise in key technologies, size of user base). Due to resources consumption and development, the resources list can be changed at each lifecycle stage. New resources might arise and they shall be considered as soon as they are available.

3.3. DS-SECO phase 3: Resources flow analysis

This phase aims at establishing a cause-effect representation for the influences among resources. This includes identifying key forces and exogenous items that interfere on resources accumulation. Key forces are internal aspects to the SECO that drive resource flows, such as investments on training and marketing expenditure. As for exogenous items, they represent factors that happen regardless of SECO participants actions, such as external competition and specific market demands. These items influence the flows by hindering or facilitating resources accumulation. A map example modelled with the Mystrategy tool is presented in Section 4.3 (Figure 2). The simulation functionality was not explored in the current version of DS-SECO model due to the lack of precise data.

3.4. DS-SECO phase 4: Dynamic Questions

External information that is not explicitly present in the resources maps can also be used for answering the dynamic questions. Considering SECO particularities, the DS-SECO model adapted the three questions originally proposed by [Warren 2002], as shown in Table 1. The **why** question focuses on explaining past performance until the analysis point defined in Phase 1. The goal of the **where** question is to focus on predictions and establish a trend in case no action is taken. Finally, the **how** question aims to find answers on how to avoid the feared performance.

Table 1. Three dynamic questions of Strategy Dynamics

	Productivity	**Robustness**	**Niche creation**
Dynamic questions	**Question 1:** Why is productivity following its current path?	**Question 1:** Why has robustness followed its current path?	**Question 1:** Why has niche creation followed its current path?
	Question 2: Where is productivity heading if the situation remain unchanged?	**Question 2:** Where is robustness heading if the situation remain unchanged?	**Question 2:** Where is niche creation heading if the situation remain unchanged?
	Question 3: How can we design a strategy to improve the performance of productivity, robustness and niche creation in the future?		

3.5. DS-SECO phase 5: Strategy assessment

If SECO health is not presenting the expected performance, strategies must be redefined to generate better outcomes in the following stages. Phase 5 then re-evaluates strategies to revert a poor performance or reinforce positive results by proposing enhancements.

4. Illustration of the DS-SECO Model: the iPhone Ecosystem

Apple has been known as an innovative firm with appealing products and loyal customers. In 2007, the iPhone SECO was born with the launch of iPhone. The main ecosystem players are Apple (keystone), device manufacturers (iPhone is manufactured by OEM), retail chains, telecom operators and independent developers as niche players.

This analysis used data published in whitepapers, IT magazines, blogs and Apple webpage as sources of evidence. All five phases of DS-SECO are applied repeatedly for each stage of SECO lifecycle. Due to space restrictions, we are presenting only a brief analysis of niche creation during self-renewal. Niche creation was chosen because this SECO is strongly dependent on players' relevant contributions to remain sustainable. Hence, we deem niche creation is the most relevant health element to demonstrate. We highlight that productivity and robustness must be equally analysed in every stage. The full application of the DS-SECO model is available at http://tinyurl.com/nw38a6c.

We consider that the iPhone SECO is in the self-renewal stage since 2010, with the development of iOS 4.0 [Macworld 2012]. This stage is characterised by a need for innovations to sustain SECO health, as exposed in 2.2. After its birth, the iPhone SECO was expanded and consolidated. This is evidenced by the undoubted leadership exerted by Apple in relation to niche players and by the absence of destructive competition. From this point onwards, the SECO needed to produce relevant innovations to battle rivalry in terms of hardware and software in order not to die. iPhone 5 was launched in September 2012 and represented a new cycle of the self-renewal stage, where innovation was crucial to sustain iPhone SECO success. As the first cycle of self-renewal occurred in 2010, we decided to conduct the strategic analysis from this point onwards. We believe that the reason for such decision is that self-renewal is considered a critical step during the lifecycle of a SECO, when the keystone faces tough challenges to demonstrate the feasibility of the ecosystem for partners.

4.1. Phase 1: time-path definition

Niche creation was fueled by the launch of iAd, which is an Apple's platform for advertisements on applications. In addition to it, the increasing number of downloads in Apple App Store indicated a potential market. Both iAd and soaring number of downloads tend to indicate higher probability of business opportunities for niche players. According to [Gottabemobile 2012], Apple had already reached roughly 7 billion downloads in 2010, September. One year later, this number raised to 15 billion [Statista 2015]. In case niche creation did not sustain a desired pace, users and even niche players would be attracted to competitor SECOs and that would severely influence SECO chances of survival, as users and customers might be attracted to the competition as well.

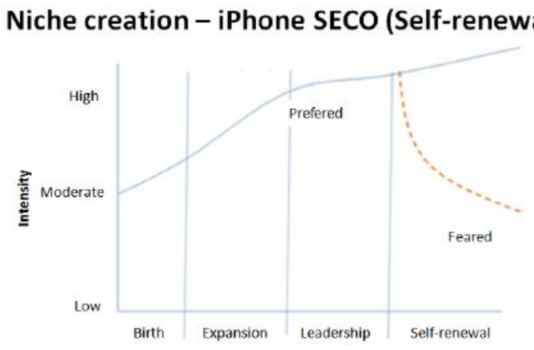


Figure 1. Time-path graph for niche creation during self-renewal

As precise numeric data was not available, the vertical axis of the graph in Figure 1 uses a generic scale for indicating intensity instead of a mathematical one. Niche creation was moderate when this SECO was born, as iPhone was a disruptive innovation at the time of its launch. Otherwise, it would be hard to attract niche players in case they saw no business opportunity. It kept stepping up in the next stages and remained stable during leadership, as this stage focuses on solving internal disputes and consolidating the SECO. To sustain this SECO during self-renewal, motivated niche players are needed. In case they leave the SECO and join competitor ones, the feared performance (i.e. a considerable drop in niche creation level) would be disastrous and that must be avoided.

4.2. Phase 2: resources identification

Potential user base increased, partially driven by the release of new iPhone and iOS versions that enhanced iPhone with new functionalities. However, the stronger threat of the Android SECO affected the growth pace for user base. Expertise on the mobile market increased as Apple consolidated its understanding of several IT consumer markets. Finally, high sales of iPhone and revenues from Apple App Store kept increasing the capital available for investment in innovation.

4.3. Phase 3: resources flow analysis

Investments in marketing and R&D remained high. Market differentials of iPhone relied on iOS and hardware enhancements. Mutualistic partnerships became even more intense, because Apple was able to avoid destructive internal disputes since leadership stage. Although Apple customers remained loyal, a stronger external competition caused some reduction in the user base flow. Investments in R&D led to innovations such as iCloud, iAd, as well as hardware and software improvements. This situation attracted potential partners to the ecosystem. External competition did not influence severely the user base flow. Figure 2 presents the consolidation of niche creation measurement in the form of a resources map. In this resources map, the square elements (capital and potential user base) are resources, while texted elements (e.g. customer loyalty, external competition) are either exogenous items or key forces. Resources influence key forces, which have a positive effect on the user base flow. On the other hand, exogenous items are acting negatively. However, the balance is positive, since potential user base is accumulating and consequently increasing niche creation.

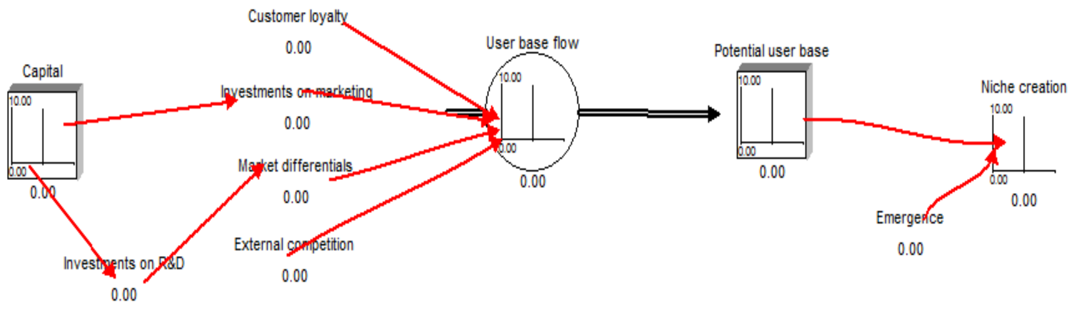


Figure 2. A resources map for niche creation during self-renewal

4.4. Phase 4: dynamic questions

New products such as iAd, iCloud and the increasing number of downloads in Apple App Store acted as ***niche creation*** drivers (why). In spite of the threat posed by Android, the iPhone SECO provided attractive financial opportunities for ecosystem participants due to the size of user base (where). Apple needed to make the iPhone SECO seem more attractive when compared to competing mobile ecosystems, so that independent developers did not abandon the SECO (how).

4.5. Phase 5: strategy assessment

Based on the outcomes of Phase 4, the main reasons to keep considering the iPhone SECO as a mobile market leader are users' passion for Apple products, frequent innovations in iOS operating system and devices design. They enabled iPhone to remain competitive and attractive. Another positive strategic outcome was Apple's ability to orchestrate the SECO in a way that participants remained active by receiving financial advantages. Finally, the attraction of customers that used products from different ecosystems leaded by Apple (e.g. MacBook, iPad, iPod) enabled the company to enhance profitability. To summarise, strategies adopted by the iPhone SECO indicate it is on the right track to ensure sustainable performance in the next cycles of self-renewal.

According to [Apple Culture 2012], Apple earned \$24,4 billion with iPhone in the last quarter of 2011, while Microsoft earned \$ 20,9 billion with Xbox, Windows Phone and Microsoft Office altogether. This shows that Apple properly managed to create a very lucrative ecosystem around the iPhone. The figures presented the results of effective strategies and can be reflected in very good SECO health indicators. However, new cycles of self-renewal need to consider the growth of the Android SECO. This ecosystem has Google as a keystone and involves players such as Motorola and Samsung. In addition, the Windows Phone SECO must also be monitored, as Microsoft acquired Nokia and by doing so, strengthened its position on the mobile market. Therefore, the DS-SECO model shall be applied on upcoming cycles of self-renewal to maintain the iPhone SECO healthy.

5. Conclusions and Future Work

The main contribution of this paper is the DS-SECO model, which may help players to analyse and select strategies during the ecosystems lifecycle. The model considers concepts from biological ecosystems and principles from Strategy Dynamics. The association of knowledge from these fields brings originality and a well stablished basis to our proposal. To illustrate the application of DS-SECO, we conducted an analysis of the iPhone ecosystem. Unfortunately, it was not possible to conduct interviews with Apple due to difficulties to formally interact with the company. We then acknowledge that our study faces limitations regarding the reliability of collected data. We aimed at illustrating how the DS-SECO model can be used in practice. The current study provided a historical analysis of strategies adopted by iPhone since its official birth in 2007 until the end of the first cycle of self-renewal. Therefore, we plan to conduct in-depth case studies to assess the DS-SECO model in practice. The model can be a useful approach to guide players involved in a SECO to assess and select appropriate strategies to ensure their success and overall ecosystem sustainability. Another future work involves an extensive adoption of strategy dynamics and DRSV, including the collection of real data as well followed by the use of simulation functionality provided by Mystrategy tool. This study was centred on strategic issues. We also regard a study centred on the relation between technical aspects and strategies in a SECO configuration as future work.

References

Apple (2012), http://www.apple.com/. Accessed 2015-06-01.

Apple Culture (2012), http://www.applecture.com/iphone-brings-more-income-than-all-of-the-microsoft-industries-96207. Accessed 2015-06-02.

Bosch, J. (2009) "From Software Product Lines to Software Ecosystems". In: Proc. 13th International Software Product Line Conference, pp 111-119.

Campbell, PRJ. and Ahmed, F. (2010) "A three-dimensional view of Software

Ecosystems". In: Proc Fourth European Conference on Software Architecture: Companion Volume.

GottaBeMobile (2012), http://www.gottabemobile.com/2011/06/03/ios-and-iphone-timeline-from-iphone-to-ios-5-in-5-years/. Accessed 2015-05-30

Hartigh, E., Tol, M. and Visscher, W. (2006) "The Health Measurement of a Business Ecosystem". In: Proc 6th Annual Meeting of the European Chaos and Complexity in Organisations Network.

Hartigh, E., Visscher,W., Tol, M., Salas, A. J. (2013) "Measuring the health of a business ecosystem". In: Jansen, S., Brinkkemper, S., Cusumano, M. A. (Eds.), Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry. Edward Elgar Publishing, Cheltenham, UK, pp. 221–246.

Iansiti, M. and Levien, R. (2004) "Strategy as ecology". Harvard Business Review 82(3).

Jansen, S. (2014) "Measuring the health of open source software ecosystems: Beyond the scope of project health". In: Information and Software Technology 56 (11), Elsevier, pp. 1508–1519.

Jansen, S., Brinkkemper, S. and Finkelstein, A. (2007) "Providing Transparency in the Business of Software: A Modeling Technique for Software Supply Networks". In: IFIP International Federation for Information Processing, Springer 243: 677-686.

Jansen, S., Finkelstein, A. and Brinkkemper, S. (2009a) "A sense of community: a research agenda for Software Ecosystems". In: Proc 31st International Conference on Software Engineering, pp 187-190.

Jansen, S., Brinkkemper, S. and Finkelstein, A. (2009b) "Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems". In: Proc 1st Workshop on Software Ecosystems, pp 34-48.

Macworld (2012), http://www.macworld.co.uk/ipod-itunes/news/index.cfm?newsid=3225998. Accessed 2015-05-20

Messerschmitt, D.G. and Szyperski, C. (2003) "Software Ecosystem: Understanding an Indispensable Technology and Industry." MIT Press.

Moore, J.F. (1993) "Predators and prey: a new ecology of competition". Harvard Business Review May/June: 75-86

Santos, R. and Werner, C. (2010) "Revisiting the Concept of Components in Software Engineering from a Software Ecosystem Perspective". In: Proc Fourth European Conference on Software Architecture, pp 135-142.

Statista Website (2015), www.statista.com. The Statistics Portal

Strategy Dynamics Website (2012), www.strategydynamics.com. Accessed 2015-06-08.

Warren, K. (1999) "The Dynamics of Strategy". Business Strategy Review 10(3): 1-16.

Warren, K. (2002) "Competitive Strategy Dynamics". London Business School, John Wiley & Sons LTD.

Yier, B., Lee, C.H. and Venkatraman, N. (2006) "Managing in a small world ecosystem": Lessons from the software sector. Harvard Business Review.

A Systematic Mapping on the Relations between Systems-of-Systems and Software Ecosystems

Helvio Jeronimo Junior[1], Cláudia Werner[1]

[1] PESC/COPPE –Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

{jeronimohjr, werner}@cos.ufrj.br

Abstract. *Currently, software development organizations have created and maintained their products and services with different technologies to one or more software platforms. This scenario involves different actors of one or more organizations, thus requiring attention for connectivity and dependence on technical, social and business issues. In this perspective, it has been recently suggested that topics of research as Systems-of-Systems and Software Ecosystems represent an effective way to construct large and complex software systems on top of one or more platforms, which are composed by different software products and involve different individuals, groups and organizations. However, these two topics have been separately investigated. Thus, in this paper we conduct a systematic mapping study aiming to identify the relations between these topics and support cooperative and collaborative research. The results showed that there is a relationship between Systems-of-Systems and Software Ecosystems as regards to some technical, social and business aspects.*

1. Introduction

Software stands out as an important element that can provide competitive advantages to organizations. However, currently, software development has become more challenging, as regards to: (i) the development of software-intensive and large-scale system to be used in complex domains in order to meet emergent needs of society and; (ii) the creation of software products, services and processes in collaboration with external partners of the organization. It is remarkable that the global software development crosses the organizational boundaries introducing new challenges to traditional Software Engineering (SE) processes. The concern is not a single product development but rather the development of multiple products (Campbell & Ahmed, 2010), in which different facets and the perspectives of value of its players (e.g., companies, clients, end-users, developers, service providers, supplier, manufacturer and others) should be considered. Thus, such challenges and aspects have been explored in SE due to the need for treatment of technical, economic and social issues.

Regarding the challenges in the development of complex software systems, as pointed out by Maier (1998), in the last years there had been growing interest of researches in a class of software-intensive system development, called Systems-of-Systems (SoS). These systems are heterogeneous, independent, supported by multiple platforms (technologies) and have a decentralized control. Besides, as discussed in Siemieniuch & Sinclair (2014), the collaboration and interoperability among actors, artifacts, companies and communities of these systems are extremely important. In another perspective, these aspects have been discussed in the context of Software Ecosystem (SECO), an effective way to construct large software systems on top of a software platform by composing components developed by actors internal and external to the organization developing the platform (Messerschmitt & Szyperski, 2003; Manikas & Hansen, 2013). In this context, specific treatments for technical, social and business issues are considered (Campbell & Ahmed, 2010). However, SoS and SECO are two research topics that have been still separately investigated, although they can be treated in a complementary way. This paper investigates the relationships between these two research topics by conducting a systematic mapping study complementing the research performed by Santos *et al.* (2014a). The main findings presented in the systematic mapping study reinforce and increase the scope of the discussions presented by Santos *et al* (2014a).

The remainder of this text is organized as follows. In Section 2, we present the research method, describing research questions, inclusion/exclusion criteria, sources of studies, search strategy and data extraction. Section 3 shows the results of this systematic mapping study and their analysis. Finally, in Section 4, the conclusions are presented.

2. Research Method

A systematic literature review and mapping study do not share all research procedures, however similar processes for searching are explicitly defined in the research protocol and reported as part of the outcomes (Kitchenham *et al.*, 2009). In this paper, before starting the search, a protocol was developed to define the main guidelines for conducting the study. This mapping study followed the process defined by Kitchenham and Charters (2007). Summing up, this process presents three main phases: (i) planning; (ii) execution; and (iii) reporting. As part of the review planning, we defined a protocol to detail the search strategy that includes the search string, selection criteria, and data extraction procedures. In the planning, aiming to find relevant studies that addressed SoS and SECO in the same paper, the following research questions (RQs) were established: *(RQ1) What are the main similar characteristics and differences between SoS and SECO?; (RQ2) What are the main areas studied from the perspective of SoS and SECO?; (RQ3) How can SoS benefit from business and social networks? and; (RQ4) What are the main challenges and limitations for SoS from the perspective of SECO?*

Aiming to include only studies contributing for this mapping study, two kinds of selection criteria were defined: inclusion and exclusion criteria. The inclusion criteria adopted were: (1) only studies written in English; (2) studies dealing and referencing any of the subjects related to SECO and SoS in their title, abstract or keywords, which contribute for answering one or more RQs; (3) technical reports, master and doctorate theses. The exclusion criteria were: (1) repeated studies found in different search engines (in this case, just one study was considered); (2) duplicate studies reporting similar results (in this case, only the most complete study was considered); (3) description of proceedings; and (4) inaccessible studies. Regarding the procedures about the search, in this study both electronic and manual search procedures were used. The justification for not uniquely using electronic search procedures is supported by Kitchenham's *et al*. (2009) recommendations that emphasize the use of a manual search to obtain a broader list of potential studies to review. Moreover, another reason was due to the fact that SECO and SoS represent recent research subjects; thus, a manual search brings extra confidence that more relevant studies might be found. The electronic search was conducted using the Scopus search engine. The reason for selecting this digital library is because it is an important repository for research in Computer Science area. The manual search was focused on 3 repositories, the International Workshop on Software Ecosystems (IWSECO), ACM Workshop on Software Engineering for Systems-of-Systems and the Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems. In addition, the manual search was performed on Google Scholar.

Table 1. The search string used to execute the systematic mapping

TITLE-ABS-KEY ((("software ecosystem*" OR "software supply network" OR "software vendor*" OR "software supply industry" OR "industry platform*" OR "ecosystems") AND ("system-of-systems" OR "system of systems" OR "systems-of-systems" OR systems of systems)) AND ("features" OR "feature" OR "characteristic" OR "characteristics" OR "difference" OR "differences" OR "similarity" OR "similarities" OR "relationship" OR "implication" OR "consequence" OR "significance" OR "benefits" OR "impacts" OR "business" OR "business networks" OR "social" OR "social networks" OR "area" OR "subject area" OR "subject field" OR "field"))

An approach was used to derive terms from the research questions to create the search string, therefore the strategy was: (1) derive the main search terms; (2) check the keywords from relevant studies already known; and (3) find synonyms and relevant keywords. After that, Boolean operators OR and AND were used to incorporate them into the search string. The first segment consisted of synonyms of SECO and SoS; the second was derived from the main RQ terms. Table 1 shows the final search string. Regarding the conduction phase, relevant studies was identified through stages as the following. First the string search was applied in the search engine previously mentioned. The automatic search found 71 papers, and the application of selection criteria was limited to studies’ title, abstract, and keywords. This information was read and only 15 papers were available for download. Finally, after reading them, the list was reduced to 8 papers. Regarding the manual search, the selection process was similar to the automatic search. This search found 7 papers, and from these the list was reduced to 3. Thus, in total, 11 papers were considered relevant for this systematic mapping study.

To support the extraction of data from the papers, the Zotero Standalone tool was used (https://www.zotero.org/). Some spreadsheets were created to support the process of papers selection in the stage of inclusion and exclusion criteria application. Thus, for each one of them a form was created to record details on how it answered the four research questions, by extracting pieces of relevant text. The Appendix shows the complete list of selected studies enumerated from S1 to S11. For each studies an identifier (ID) was defined, being used to reference the mapping along the text.

3. Results and Considerations

In this section, we report and discuss the answers for each RQ. Due to space limitation, only some of the main findings and their discussions are presented:

Answers to RQ1 (similar characteristics and differences)

Table 2 presents the main similar characteristics between SoS and SECO identified in the studies. A remarkable result is that many studies (S1, S2, S 4, S5, S10 and S11) have linked the similarities between SoS and SECO to the existence of multiple products (e.g., software systems) over one or more technological platforms. Besides that, some studies (S2, S3, S4, S5, S7, S8 and S9) have linked this relation to the interaction between the different actors that can exist in a SoS and SECO, and some business aspects (S3, S4, S9 and S10).

Table 2. Similar characteristics between SoS and SECO

Characteristics	Study ID
Existence of multiple products (software systems) over one or more technological platforms, in which they can operate in different environments to provide a final service	1, 2, 4, 5, 10, 11
Existence of platforms, software and artifacts developed by internal and external actors, who can have different perspectives of value	10, 11
Existence of decentralized/distributed systems	7
Many key social aspects, such as: the importance given to interaction and collaboration and communication among the different internal or external actors of the SECOs and SoS	2, 3, 4, 5, 7, 8, 9, 10, 11
Some key aspects of business, such as: the importance given to prioritization of business goals and requirements, in order to support the design and knowledge on software architecture; innovation from the involvement of players (organizations and individuals), thus, enabling the creation of an ecosystem that is more sensitive to the market trends; competitive environment, where players demand increase of connected systems and services connected, efficiency, productivity and quality, and also reduced costs, time-to-market, and delivery.	3, 4, 9, 10
Architectural stability of the platforms of a SECO can be compared to the operational independence of the SoS	1, 2

Regarding the differences, none addresses an explicit difference between SoS and SECO. Although, some studies pointed out complementary aspects and comparison between the two topics. In studies S2 and S10, the authors emphasize that SECO can be seen as an application domain for SoS. S4 mentioned that SECO provides a complementary organizational view for development of SoS. In S9 the authors pointed out that SoS is a type of SECO. For S8, the term SoS is now becoming more common as cyber ecosystems or more conveniently community ecosystems refer to systems of collaborating communities.

Answers to RQ2 (main areas)

Table 3. The main areas studied in SoS and SECOs

Areas	Study ID
Interaction and relationship among the players of a community	1, 2, 3, 4,5, 8, 10
Technical aspects linked to design of architecture software systems and/or platforms, such as: connectivity and modularity between systems and components; heterogeneity of hardware and software; geographically distributed software systems; the use of techniques to support the analysis of software architecture and; and the architecture documentation	3, 4,5,7,9,10, 11
Evolution, assessment and sustainability of the platforms and their respective software products	1, 8, 10, 11
Opening of the architecture of platforms and software products	2, 4, 6
Goals, processes and business models	1, 2, 3, 4, 9, 10
Reuse of software and components	3, 6, 11
Quality attributes (mainly: connectivity, interoperability, security, testability, stability, flexibility, robustness e integrity)	1, 2, 3, 4, 5, 9, 11
License of software and components	2, 3, 6
Innovation of products and services	5, 7

The main areas studied in SoS and SECO are presented in Table 3. A notable result is that, even if succinctly, technical, social and business aspects are discussed in the same paper. Nevertheless, the studies focus more on technical aspects linked to design of software systems and/or platforms architecture (S3, S4, S5, S7, S9, S10 and S11), communication and collaboration among the players (S1, S2, S3, S4, S5, S8 and S10), quality attributes both required in SoS as SECOs (S1, S2, S3, S4, S5, S9 and S11), and on goals, processes and business models (S1, S2, S3, S4, S9 and S10). Other studies address issues linked to evolution, assessment and sustainability of platforms, opening of platforms and software products, reuse of software and components, license of software and components and innovation of products and services.

Answers to RQ3 (benefiting from business and social networks)

Following the main benefits that business and social networks can provide to the development of SoS are presented.

Most studies (S1, S4, S5, S6, S7, S8 and S11) point out that business and social networks can enable the creation of a community between internal actors and from third parties that cooperate and share technical knowledge, in order to provide technical solutions collaboratively to support the design and sustainability of the architecture of SoS constituent systems. Other benefits found in some studies point out that providing a potential to structure collaborative business models and processes in order to consider the different perspectives of value of players (S1, S2, S3, S4 and S8), as well as opportunities to minimize and amortize costs and technical and business risks (S4, S9, S8 and S1).

In this perspective, some studies also point out that business and social networks increase possibilities to innovate products and services these systems (S1, S2, S4, S5 and S10), help analyze demands, strategies of marketing and production (S2, S4 and 8), and enable a better understanding of its constituent systems architecture (e.g., its product line) to support reuse of software and components (S6 and S11).The social and business networks can provide many benefits to the SoS development. These are similar to those addressed by Santos *et al.* (2014b) in the SECO context, as for example: (i) the visibility and mapping of connections between people, or between people and organizations - it is possible to access the knowledge of members in a network, and sometimes their contacts; (ii) the bigger power of propagation on products and services; and (iii) new market niches and the trading of new products. Thus, the benefits found in these studies are associated to the three dimensions (i.e., technical, social and business) considered in SECO.

Answers to RQ4 (main challenges and limitations)

Through the studies (S1, S2, S4, S5, S7, S9 and S10) it was observed that the establishment of efficient organizational strategies and business models with partners is one the main challenge and limitation for SoS from the perspective that involves a partner community (SECO). It was also observed that business aspects in SoS still seem to be a great challenge.

In addition, several technical concerns were observed, such as: assessment, evolution and stability of the software architectures of the SoS and platform interfaces (S1, S2, S3, S7, S9 and S11); compliance with quality attributes (S2, S3, S4, S9 and S11), which are required in the context SoS (Santos *et al.*, 2015); lack of tools and adequate technical infrastructure to support technical decision-making, perform verification and validation activities and, the evolution of the platforms architecture and/or its software systems (S1, S4 and S5); lack of tools and adequate technical infrastructure to support the interaction and communication of a collaborative community (S8); opening of SoS platforms and/or of its constituent systems (S1, S2, S4 and S6), in which one of the concerns should be security, as discussed by Barbosa *et al.* (2013) in the SECO context, by opening its architecture, a software application might suffer attacks that operate from inside or outside the organization; difficulty in reuse (S3, S6 and S11), because as pointed out by Botterweck (2013) an approach for variability management and systematic reuse in SoS is required and in this sense the Product Line Engineering (PLE) is relevant and helpful, however generally there is little discussion in the PLE literature regarding SoS. In another perspective, Werner (2009) emphasizes that well-known software reuse approaches such as Component-Based Development and Software Product Line can lead companies to SECO. In this direction, we believe that if software components are reused more widely in the SoS context, some of the reuse benefits can be achieved, such as: increased reliability, reduced costs and potentially increased agility in evolving to meet the emergent behavior of these systems. But, in this case challenge is to realize the benefits of this approach when individual components are heterogeneously licensed (S1 and S6), in which each potentially with a different license, rather than with a single license (Scacchi & Alspaugh, 2012) as in SoS. Another challenge pointed out was the heterogeneity of ecosystems, platforms and its constituent systems (S1), in which we can consider as the responsible for all challenges and limitations described above.

4. Conclusion

As previously discussed, it is observed that the solutions for SECO and SoS are still individually proposed by isolated teams in order to meet particular domain-oriented problems. But, as identified in this mapping study there are similar characteristics and complementary aspects between SoS and SECO, showing that there are opportunities for cooperative and collaborative research between these two topics.

The findings presented in this systematic mapping study reinforce and increase the scope of the discussions presented by Santos *et al.* (2014a), since it was possible to more clearly identify the relationships between these two topics. As can be seen, our results showed that the relationships between them are associated to specific issues of the three dimensions of a SECO. In addition, we identified the main areas studied within this context, the possible benefits that a SoS can achieve from business and social networks, and the main challenges and limitations. This systematic mapping study pointed out that most areas studied in this context are linked to technical aspects. As such, according to Klein & McGregor (2013), the concept of architecture has been amplified to the so-called SoS or industry platform, in order to help the comprehension of architecture in SECO. This kind of platform provides support to a set of systems that need to interact to form a SoS (Maier, 1998). These are complex, interdisciplinary systems whose functionalities and purposes can dynamically evolve, encompassing several new challenges to be developed.

In this sense, the concepts of virtual and collaborative SoS (DoD, 2008) have been discussed in the SECO context, allowing collaboration of different constituent systems and organizations to produce emergent functionalities. In both collaborative and virtual SoS, SECO is more valuable because in these categories there is no strict control over the constituent systems. In a perspective of social and business issues, a SECO provides a complementary organizational view to SoS development, which introduces roles and rules of interaction, collaboration and synergistic capabilities for its constituent systems. From this discussion it is possible to confirm the existence of many similarities between SoS characteristics (Maier, 1998) and SECO technical challenges (Bosch, 2010), which were raised in the research performed by Santos *et al*. (2014a). The operational independence of constituent systems of a SoS can be compared to architectural stability required for SECO platforms as regards to their components, services, and applications. In this case, the strategies of software systems integration and component-based development can be combined to support application programming interface issues. The platform evolution directly depends on the SECO community's emerging requirements and contributions, as well as the adjustments of underlying hybrid business models. It requires explicit modelling of roles in different organizations and the rules that govern their internal and external interactions with respect to each organization, for instance, when an organization collaborates with independent third-party. Thus, SoS evolutionary development should also take into account the business and social issues, and not only the environment's technical issues. In this sense, SoS architecture models should be extended to deal with context variables based on value chains and social networks. In turn, emergent behavior produced by constituent systems of a SoS working together can be linked to the security and reliability in SECO.

Regarding the treats to validity of this study, the main can be associated to: (i) an eventual omission of studies and bias in the extraction data; (ii) the loss of relevant studies due to the lack of agreed terminology for SoS and SECO; (iii) the possible existence of relevant studies that do not mention the keywords that were chosen and; (iv) the number of electronic used databases, since only the Scopus search engine was used in this systematic mapping study. Regarding to bias in the data extraction, some difficulties were faced to extract useful information from the studies found, since many did not explicitly answer the research questions. As future studies, we aim to extend this study by conducting searches on other search engines and, if necessary, perform adjustments in the search string to find more relevant studies, probably addressing other issues that have been discussed by literature, but were not found in this systematic mapping study (e.g., health of SoS-Ecosystem). Moreover, we intend to investigate how SECO platforms can benefit from SoS mindset, which was not in the aim of this study.

References

Barbosa, O. et al. (2013) "A Systematic Mapping Study on Software Ecosystems through a Three dimensional Perspective". In: Jansen, S., Cusumano, M. & Brinkkemper, S. (eds.) Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry. Edward Elgar Publishing, pp. 59 81.

Bosch, J. (2010) "Architecture Challenges for Software Ecosystems". In: Proc. of the 4th European Conference on Software Architecture (ECSA), 2nd International Workshop on Software Ecosystems (IWSECO), Copenhagen, Denmark, pp. 93-95.

Botterweck, G. (2013). "Variability and evolution in systems of systems". In: EPTCS, vol. 133, pp. 8-23

Budgen, D. *et al*. (2008) “Using Mapping Studies in Software Engineering”. In: Proc. of Psychology of Programming Interest Group Workshop, Lancaster, pp. 95–204.

Campbell, P. R. J., Ahmed, F. (2010) “A Three-dimensional View of Software Ecosystems”. In: 4th ECSA, 2nd IWSECO, Copenhagen, Denmark, pp. 81-84.

DoD (2008) “System engineering guide for systems of systems”. Technical Report, Version 1.0, August 2008.

Kitchenham, B., S, Charters. (2007) “Guidelines for performing systematic literature reviews in software engineering”. Technical Report, Keele University and Durham University Joint Report.

Kitchenham, B. et al. (2009) “Systematic Literature Reviews in Software Engineering – A Systematic Literature Review”. Information and Software Technology, vol. 51, n. 1, pp. 7–15.

Klein, J., McGregor, J. (2013) “System-of-Systems Platform Scoping”. In: 4th International Workshop on Product Line Approaches in Software Engineering (PLEASE), San Francisco, USA, pp. 1-4.

Maier, M. (1998) “Architecting principles for systems-of-systems”. System Engineering, vol. 1, n. 4, pp. 267-284.

Manikas, K., Hansen., K. (2013) “Software Ecosystems – A Systematic Literature Review”. Journal of Systems and Software, vol. 86, n. 5, pp. 1294-1306.

Messerschmitt, D., Szyperski, C. (2003) “Software Ecosystem: Understanding an Indispensable Technology and Industry”. The MIT Press.

Santos, D. S. *et al*. (2015) “An Investigation on quality attributes of Systems-of-Systems”. In: Technical report, SP, Brazil.

Santos, R. *et al*. (2014) “On the Relations between Systems-of-Systems and Software Ecosystems”. In: WDES 2014, pp. 58-62.

Santos, R. P. *et al*. (2014b) “Using Social Networks to Support Software Ecosystems Comprehension and Evolution”. Social Networking, vol. 3, n. 2, pp. 108-118.

Scacchi, W., Alspaugh, T. A. (2012) “Understanding the role of licenses and evolution in open architecture software ecosystems”. Journal of Systems and Software, vol. 85, n. 7, pp. 1479-1494.

Siemieniuch, C. E.,Sinclair, M. A. (2014). “Extending systems ergonomics thinking to accommodate the socio-technical issues of Systems of Systems”. Applied ergonomics, vol. 45, n. 1, pp. 85-98.

Werner, C. (2009). “Building Software Ecosystems from a Reuse Perspective”. In: Proc. of the First International Workshop on Software Ecosystems, 11th International Conference on Software Reuse, Falls Church, VA, USA, p. 3.

Appendix: The selected studies

ID	Reference
S1	Da Silva Amorim, S. *et al*. (2014) “When Ecosystems Collide: Making Systems of Systems Work.” In: Proceedings of the 2014 European Conference on Software Architecture Workshops. ACM, p. 29.
S2	Axelsson, J. *et al*. (2014) “Characteristics of software ecosystems for Federated Embedded Systems: A case study”. Information and Software Technology, v. 56.11, pp.1457-1475.
S3	Lutz, M. *et al*. (2014) "Service Robot Control Architectures for Flexible and Robust Real-World Task Execution: Best Practices and Patterns." In: Workshop Roboter-Kontrollarchitekturen.
S4	Papatheocharous, E. *et al.* (2013) “Issues and challenges in ecosystems for federated embedded systems.” In: Proc. of the First International Workshop on Software Engineering for Systems-of-Systems (SESoS). ACM, pp. 21-24.
S5	Delicato, F. C. *et al*. (2013) “Towards an IoT ecosystem” In: Proc. of the First International SESoS. ACM, pp. 25-28.
S6	Mattmann, C. *et al*. (2012) “Developing an open source strategy for NASA earth science data systems.” In: IEEE 13th International Conference on Information Reuse and Integration, pp. 687-693.
S7	Chen, H. M., Kazman, R. (2012) “Architecting ultra-large-scale green information systems." In: Green and Sustainable Software (GREENS), 2012 First International Workshop on. IEEE, pp. 69-75.
S8	Hawryszkiewycz, I. T. (2011) “Open Modeling For Designing Community Ecosystems." In: https://opus.lib.uts.edu.au/research/handle/10453/19216. Last accessed on: Jun, 2015.
S9	Kazman, R. *et al*. (2012) “Scaling up software architecture analysis". Journal of Systems and Software, 85.7, pp. 1511-1519.
S10	Santos, R. *et al*. (2014) "On the Relations between Systems-of-Systems and Software Ecosystems." In: WDES 2014, pp. 58-62.
S11	Lytra, I. *et al*. (2015) "Reusable Architectural Decision Models for Quality-driven Decision Support: A Case Study from a Smart Cities Software Ecosystem. In: 3th International SESoS.

Oportunizando a Aprendizagem e Gerência do Conhecimento em Ecossistemas de Software

Davi Viana[1,2] e Tayana Conte[1]

[1]Grupo de Usabilidade e Engenharia de Software (USES) – Universidade Federal do Amazonas (UFAM) – Manaus – AM – Brasil

[2]Fundação Centro de Análise, Pesquisa e Inovação Tecnológica (FUCAPI) Manaus – AM – Brasil

{davi.viana,tayana}@icomp.ufam.edu.br

Abstract. *Software development activities are knowledge intensive in many contexts. Within the Software Ecosystems context, it is necessary to guarantee that the different actors have the necessary knowledge to carry out their activities. The many interactions that occur between the actors and the artifacts in Software Ecosystems can enable the application of Learning and Knowledge Management strategies. This paper aims at presenting and discussing these possible opportunities. As a result, we intend to encourage future research involving Learning and Knowledge Management in the context of Software Ecosystems.*

Resumo. *Atividades de Desenvolvimento de Software em diversos contextos são intensas em conhecimento. No contexto de Ecossistemas de Software é necessário garantir que os diversos atores tenham os conhecimentos necessários para a execução de suas atividades. As diversas interações que ocorrem entre atores e artefatos em Ecossistema de Software podem oportunizar a aplicação de estratégias de Aprendizagem e Gerência do Conhecimento. Este artigo busca apresentar e discutir essas possíveis oportunidades e, assim, viabilizar futuras pesquisas envolvendo Aprendizagem e Gerência de Conhecimento no contexto de Ecossistemas de Software.*

1. Introdução

A indústria de software é fortemente baseada em conhecimento e aplicar as boas práticas de Engenharia de Software (ES) envolve uma atividade intensa de conhecimento (Levy e Hazzan, 2009). Esse conhecimento se tornou uma vantagem competitiva e sustentável para as organizações, visto que: o mercado, produtos, tecnologias e a própria sociedade mudam de forma bastante ágil (Ruhe, 2001). Observa-se que uma dessas mudanças está ocorrendo nas formas de desenvolvimento de software. Os Ecossistemas de Software (ECOS) vem emergindo como alternativa para o desenvolvimento de diversos projetos em torno de uma tecnologia de software central, onde projetos são desenvolvidos de maneira global, com o envolvimento e colaboração constante de atores externos (Santos *et al.*, 2014; Manikas e Hansen, 2013). Essa alternativa faz surgir novos desafios de engenharia frente às diferentes especificidades já existentes na Engenharia de Software (como reuso de software, colaboração e entre outros) (Seichter *et al.*, 2010). Isto ocorre, pois o desenvolvimento dos produtos de software não acontece em torno de um projeto, mas sim na interação de uma rede de diversos atores e artefatos.

A interação em ECOS pode representar um desafio para este contexto, pois é necessário atentar para questões geográficas e culturais, assim como, garantir que os conhecimentos sobre a plataforma e tecnologias em desenvolvimento sejam transmitidos e aprendidos pelos atores. Segundo Menolli *et al.* (2015), diferentes abordagens e estratégias de Aprendizagem e Gerência do Conhecimento (GC) são utilizadas no contexto de desenvolvimento de software tradicional. Essas estratégias possuem o objetivo de coletar, tratar e compartilhar os conhecimentos relevantes (Schneider, 2009). Contudo, é necessário investigar como as interações em ECOS oportunizam a aplicação dessas abordagens e estratégias e os benefícios que elas podem trazer para o desenvolvimento de software em ECOS.

Este artigo apresenta discussões iniciais sobre possibilidades de pesquisa relacionadas à Aprendizagem e GC em interações entre atores e artefatos no contexto de ECOS. Além desta seção, este artigo está estruturado da seguinte forma: a Seção 2 descreve conceitos tradicionais de Aprendizagem e GC; a Seção 3 apresenta uma breve discussão sobre possibilidades de viabilizar a Aprendizagem e GC em interações entre atores e artefatos de ECOS; e, a Seção 4 apresenta as considerações finais deste artigo.

2. Aprendizagem e Gerência do Conhecimento Tradicional e em ECOS

Para Nonaka e Takeuchi (1995), o conhecimento sempre se origina nas pessoas sendo criado através da interação entre o Conhecimento Tácito e o Explícito. Essa interação deu origem ao modelo SECI (do inglês, *Socialization, Externalization, Combination e Internalization*). Cada processo deste modelo representa uma conversão de conhecimento. Durante o processo de socialização, o Conhecimento Tácito é compartilhado diretamente com outra pessoa. No processo de externalização, o Conhecimento Tácito é convertido em Explícito. Já na etapa de combinação ocorre a agregação de componentes isolados do Conhecimento Explícito para a geração de um novo Conhecimento Explícito. À medida que um novo Conhecimento Explícito é compartilhado, outras pessoas podem internalizá-lo, criando Conhecimento Tácito.

Ao gerenciar o conhecimento, as organizações podem reagir melhor às demandas de clientes e mercados (Schneider, 2009). Além do tratamento provido pela GC, é necessário estimular a aprendizagem dos conhecimentos necessários à execução de atividades. Abordagens de Aprendizagem buscam disseminar o conhecimento em organizações de desenvolvimento de software (Ruhe, 2001). A aprendizagem deve ocorrer por meio de aplicação das habilidades de criação, transferência e absorção de conhecimentos. Uma das formas de verificar se a Aprendizagem está sendo efetiva é através da verificação de mudança de comportamento nos indivíduos durante a execução de suas atividades.

A Gerência do Conhecimento pode ser vista como um desafio de pesquisa em ECOS. Santos et al. (2014) categorizam os principais desafios e oportunidades de pesquisa nesta área. Entre essas categorias, podem-se destacar os Aspectos Sociais e Gerência de Conhecimento. Os conhecimentos gerados e utilizados no ECOS devem ser aproveitados adequadamente pelos projetos e pela plataforma.

Visando apoiar a GC em ECOS, Seichter *et al*. (2010) propõem uma abordagem para tratar os artefatos de software gerados e necessários em um ECOS. Essa abordagem utiliza o conceito de análise de redes sociais para auxiliar na GC dos artefatos de um ECOS. Como resultados, os autores apresentam um conjunto de relacionamentos (ou interações) entre artefatos e atores, além de apresentar discussões sobre cenários de aplicações da abordagem proposta. Essas interações são utilizadas neste presente trabalho para explorar as oportunidades de Aprendizagem e GC, visando oportunizar pesquisas na área de ECOS.

3. Interações em ECOS e as oportunidades em Aprendizagem e Gerência do Conhecimento

As interações podem auxiliar na transferência de informações entre os diversos atores em ECOS (Seichter *et al*., 2010). As interações auxiliam a determinar o quão consistente é um ecossistema (McGregor e Amorim, 2014). A Tabela 1 apresenta a definição dessas interações. Essas interações levam em consideração a rede social existente entre atores e artefatos em ECOS.

Tabela 1. Tipos de Interações em ECOS (Seichter *et al*., 2010)

Interação	Descrição
Ator → Ator	Como toda rede social, atores podem interagir com atores, utilizando troca de mensagem através ferramentas de apoio à comunicação. Por exemplo: os atores podem usar um mural de comentários para trocar informações.
Artefato → Ator	Todos os atores, que estão conectados a um determinado artefato na rede social, são informados sobre uma mudança de status desse artefato. Repositórios de código e ferramentas de *bug tracking* podem auxiliar nessa comunicação.
Ator → Artefato	Os atores podem avaliar ou comentar as atualizações dos artefatos. Essa interação pode ocorrer com o objetivo de enriquecer as informações/conhecimentos do artefato. Por exemplo: comentar ou avaliar a mudança de interface de um componente.
Artefato → Artefato	Um artefato pode enviar mensagens automáticas para outros artefatos. Por exemplo: um artefato A sofreu uma determinada alteração. Após o *commit* dessas alterações neste artefato A, todos os outros artefatos dependentes deste artefato A podem ser automaticamente alterados também.

Em ECOS, a interação “Ator → Ator” normalmente ocorre através da utilização de ferramentas de comunicação. O ator, que precisa enviar ou registrar informações para serem utilizadas por outros atores, pode utilizar técnicas de externalização do conhecimento. Essas técnicas buscam estruturar as informações que são compartilhadas. Para isso, pode-se utilizar abordagens de codificação do conhecimento (Rabelo *et al*., 2014; Jeung-Tai e Chihui, 2009). Em contrapartida, o ator que recebe/utiliza as informações precisa ser motivado a internalizar os conhecimentos externalizados através dos canais de comunicação.

Os artefatos do ECOS podem ser considerados pelos atores como repositórios de conhecimento. Na interação “Artefato → Ator”, ao disseminar os artefatos pode ser viabilizado a disseminação do conhecimento relevante para o desenvolvimento das tecnologias e evolução da plataforma. Já na interação “Ator → Artefato”, os atores precisam ser motivados a disseminar o conhecimento nos artefatos gerados no ECOS. Através dos comentários e/ou avaliações realizadas nos artefatos, os atores podem compartilhar conhecimentos relevantes para que outros atores possam utilizar esses artefatos adequadamente.

Por fim, a interação “Artefato → Artefato” pode oportunizar a aplicação de combinação de conhecimentos explícitos. Quando os artefatos dependentes recebem atualizações automáticas a partir de uma atualização de um artefato, o conhecimento descrito anteriormente também pode ser atualizado, desta forma é possível agregar mais conhecimento explícito nos artefatos, gerando assim novos conhecimentos.

4. Considerações Finais

O conhecimento que é compartilhado nas interações entre atores e artefatos em ECOS pode auxiliar no desenvolvimento e manutenção das tecnologias de software. Em ECOS, aplicar estratégias de Aprendizagem e GC tende a ser mais crítico, devido às características inerentes deste contexto. Desta forma, é preciso oportunizar a aplicação dessas estratégias durante as interações em ECOS. Como próximos passos, verifica-se a necessidade de analisar mais a fundo cada tipo de interação para relacioná-las às soluções já existentes em Aprendizagem e GC. Essa análise será realizada através da execução de Estudos de Caso da aplicação de determinadas estratégias de Aprendizagem e GC em ECOS. Desta forma, será possível buscar possíveis evoluções dessas estratégias com o objetivo de torná-las mais adequadas ao contexto de ECOS.

Agradecimentos

Os autores agradecem ao apoio da FAPEAM através do processo 062.00578/2014.

Referências

Jeung-Tai, T e Chihui, C. (2009) “Organizational Knowledge Sharing Through Mind Mapping”. In: 6th International Conference on Fuzzy Systems and Knowledge Discovery, vol. 2, pp. 305-309.

Levy, M., Hazzan, O. (2009). “Knowledge management in practice: The case of agile software development”. In: ICSE Workshop on Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09, Vancouver, pp. 60-65

Manikas, K., Hansen, K. (2013) “Software Ecosystems – A Systematic Literature Review”. In *Journal of Systems and Software*. 86(5):1294-1306.

McGregor, J., Amorim, S. (2014). “Ecosystem Business Models and Architectures.” In: Anais do VIII WDES, Maceió, pp. 33-40.

Menolli, A. L., Cunha, M. A., Reinehr, S., Malucelli, A. (2015). ““Old” theories, “New” Technologies: Understanding knowledge sharing and learning in Brazilian software development companies”. In: J. *of Information and Software Technology*. 2015(58): 289-303.

Nonaka, I., Takeuchi, H., 1995, The Knowledge-Creating Company, 17th ed. Oxford Univ. Press.

Rabelo, J., Viana, D., Santos, G., Conte, T. (2014). “Usando PABC-Pattern para Codificar o Conhecimento: Um estudo Experimenta”l. In: Proceedings of XIII Simpósio Brasileiro de Qualidade de Software (SBQS 2014). Blumenau, pp. 1-15.

Ruhe, G., 2001, “Learning Software Organisations", Handbook of Software Engineering and Knowledge Engineering” (S.K. Chang, ed.), World Scientific Publishing, 2001.

Santos, R., Valença, G., Viana, D., Estácio, B., Fontão, A., Marczak, S., Werner, C., Alves, C., Conte, T., Prikladnicki, R. (2014). “Qualidade em Ecossistemas de Software: Desafios e Oportunidades de Pesquisa”. In: Anais do VIII WDES, Maceió, pp. 41-44.

Schneider, K., (2009). Experience and Knowledge Management in Software Engineering Heidelberg, Springer.

Seichter, D., Dhungana, D., Pleuss, A., and Hauptmann, B. (2010). “Knowledge Management in Software Ecosystems: Software Artefacts as First-class Citizens”. In: Proceedings of the 4th ECSA, 2nd IWSECO, Copenhagen, Denmark, pp. 119-126.

Observing the health of the ecosystem supporting the emerging connected vehicle system of systems

John D. McGregor[1], **Simone da Silva Amorim**[2], **Eduardo Santana de Almeida**[3,4], **Christina von Flach G. Chavez**[4]

[1] Clemson University, South Carolina, USA

[2]Federal Institute of Education, Science and Technology of Bahia, Bahia, Brazil

[3]Fraunhofer Project Center for Software & Systems Engineering, Bahia, Brazil

[4]Federal University of Bahia, Bahia, Brazil

simone.amorim@ifba.edu.br, johnmc@clemson.edu, {esa,flach}@dcc.ufba.br

Abstract. *A healthy ecosystem creates new features and products and attracts new participants. In this study a set of accepted criteria for a healthy software ecosystem is investigated as to whether those criteria adequately address the health of the more complex ecosystem supporting a system-of-systems. The system-of-systems composed of intelligent traffic control infrastructure, connected vehicles, and people is supported by an ecosystem that is very healthy due to government and industry support. We use this socio-technical ecosystem as the testbed for ecosystem health metrics. This preliminary work will guide a more in-depth study considering other criteria.*

1. Introduction

Today's vehicles are increasingly connected to intelligent traffic infrastructure, the Internet, and to each other. In many cases these connections are the result of an integration of systems that are independently owned and operated. Smart phones are connected via Bluetooth radio to the navigation system of the car. New opportunities are rapidly emerging. DSRC radios will provide vehicle to vehicle (V2V) coordination for platooning sets of vehicles for fuel efficient travel effectively creating a new system. This evolution is integrating the efforts from a variety of communities in new ways. The resulting ecosystem is an interaction of the ecosystems surrounding these diverse domains.

The connected vehicle traffic control system is a system-of-systems (SoS) according to the definition given by Maier [Maier 1998]. A standalone system joins the SoS when it comes within radio range or connects through a network, and then leaves the system as it moves away, or disconnects, from the other elements of the system. These independently owned and operated systems must interoperate for the traffic control system to achieve its purpose, i.e. the safe and efficient management of traffic flow.

As described in our previous work, each of the standalone systems composed into the system of systems is nurtured by its own socio-technical ecosystem [Amorim et al. 2014]. A socio-technical ecosystem gives equal weight to the people and the technical issues involved in creating and sustaining a product [Feiler et al. 2006]. The ecosystem consists of the organizations that cooperate and compete with the organization producing the system. The actions of the ecosystem are also the result of actions and culture of the people working in those organizations.

The SoS resulting from the composition of the standalone systems is likewise supported by an ecosystem. In previous work we described how the ecosystems, from which the systems in the SoS come, could be used to guide the assembly of the SoS [Klein and McGregor 2013]. The SoS can be successful only if its constituent systems have business strategies and technical roadmaps that are sufficiently aligned. The rest of this paper will refer to the set of constituent systems that are integrated to form the SoS and to the constituent ecosystems that interact to form the ecosystem of the SoS.

A prime concern, as the SoS ecosystem is being formed, is whether the ecosystem is healthy and will remain so. Our hypothesis is that the health of this new ecosystem is related to the health of the ecosystems containing the constituent systems. In this short paper we will expand on what that means, give examples from the connected vehicle domain, and describe challenges to continue this work.

Several models for a SoS are possible. Due to space limitations, we will address only one: a platform-based SoS, in which all of the systems are developed by independent but collaborating organizations [Klein and McGregor 2013]. The Maier criteria call for operational and managerial independence of each system in the SoS. In the platform-based approach, systems are managerially and operationally independent but are designed on a shared platform of common services. The shared platform provides services needed by two or more of the constituent systems.

The rest of this paper is structured as follows: section 2 defines what we mean by the health of a SoS ecosystem; section 3 expands on the connected vehicle example; section 4 describes some challenges to keeping the SoS ecosystem's health;and finally section 5 presents conclusions and future work.

2. Health of the SoS Ecosystem

The organizations in the ecosystem surrounding a SoS are responsible for assembling the SoS and for maintaining it in the face of continually evolving constituent systems. There is a dependency between the SoS ecosystem and those of the standalone systems. The SoS must anticipate updates to the constituent systems and will feedback bug reports and change requests to the appropriate ecosystems. Since the ecosystem strategy is intended to contribute to the health of an organization employing the strategy, we are interested in the health of this new ecosystem and the confounding effects, if any, of these inter-ecosystem dependencies. At a minimum we can use the same metrics by which we evaluate the health of any software ecosystem: Productivity, Niche Creation, and Robustness [Iansiti and Levien 2000]. In a SoS's context these may take on somewhat different meanings.

The productivity of the ecosystem surrounding a SoS is tied to the productivity of the ecosystems surrounding the constituent systems. Consider a request for a new feature in the SoS. The SoS organization is responsible for parsing the change request and determining how the new feature would be provided. This will often result in change requests being created for several of the constituent systems. Some of the change requests will be directed at the platform team and others at the constituent systems. In both cases there are dependencies among these changes. That is, some changes are needed before other changes can be created. The SoS's change control board identifies the dependencies and addresses them in submitting the requests to the other organizations.

The SoS ecosystem must encourage new collaborators with new ideas to establish new products and new markets. Much of this new activity will occur in niches within existing markets. New business models, new technologies, and changing circumstances such as aging populations lead to new ideas. Typically a SoS ecosystem is not as agile as a single system but niche creation actually often begins in the ecosystem surrounding one of the constituent systems. If the new niche is first recognized in an individual system's ecosystem, the market's reception can be judged with less risk than at the SoS level niche.

SoS ecosystems must be robust even with organizations entering and leaving the ecosystem, priorities changing, and popularity trending up and down. One approach to this is redundancy in suppliers - multiple ways of providing a specific feature. In the SoS ecosystem with relationships to several different ecosystems, it is often possible to identify multiple sources. Another approach is to reduce the cost of entry into the new market. A platform-based SoS will be more robust than a single system ecosystem since to enter a platform-based ecosystem an organization only needs to build from the platform up rather than from scratch.

Our initial investigation identified an additional health characteristic for a SoS ecosystem - cohesiveness. The constituent systems in an SoS must fit well together. Cohesiveness can be seen in how much glue code is needed between the systems within the ecosystem. Over time the individual systems evolve and may require additional glue code to interoperate. A cohesive ecosystem has minimum amount of glue code.

3. Example

Connecting vehicles to the traffic control infrastructure and to other vehicles holds the promise of significant improvements in safety and fuel efficiency. The United States Department of Transportation (DoT) has created the Connected Vehicle Reference Implementation Architecture (CVRIA). This reference architecture is the blueprint for a system of systems for connected vehicles.

A connected vehicle has wireless connections to other devices. It may use a cellular, Bluetooth, DSRC[1], WiFi, or other type of communication protocol. All of the onboard devices interface with various services in the vehicle and connect to different types of servers to which the device can connect at a point in time. At any instant in time the SoS is composed of interacting traffic infrastructure, vehicles, and people.

Productivity - The CVRIA ecosystem is currently very productive. There is a large number of infrastructure products designed to be compatible with the CVRIA. The DoT recently had a funding program related to CVRIA compliant products and this is encouraging further development.

Niche creation - The CVRIA ecosystem is creating a number of new research initiatives in the area of safety of connected vehicles. Several of the V2V technologies are creating new opportunities.

Robustness - Government funding as well as support from a number of original equipment manufacturers makes the connected vehicle traffic ecosystem very robust. The promise of greatly reduced accidents is attracting much participation in the development

[1]Dedicated Short-Range Communications

of products that will interoperate within the ecosystem. Government ownership of much of the traffic infrastructure ensures that the ecosystem will remain robust further encouraging activity.

Cohesiveness - Existing traffic regulations, constraints described in the CVRIA result, and Society for Automotive Engineers (SAE) standards ensure the cohesiveness of this ecosystem.

4. Challenges

We have observed that there are several challenges to achieving adequate health of the constituent ecosystems:

Community alignment - The challenge is the management of different rules and behavior standards among the groups in the individual ecosystem communities. The rules of a single ecosystem should be aligned to avoid a collision of interest among the members of the other communities.

Management of multiple markets - The ecosystems should support the SoS and other applications and be present in different markets inside and outside the SoS. The directions of the SoS and ecosystems should be synchronized to avoid the withdrawal of participant systems.

Architectural decisions - For a platform-based SoS, architectural decisions must be separated into those that support all applications on the system-of-systems and those that support only those applications within a single ecosystem. Besides, another challenge is to manage all dependencies among these projects to satisfy both niche markets.

5. Conclusions and Future work

Our initial investigation of ecosystems that support SoS has identified one new characteristic of ecosystem health - cohesiveness. This characteristic reflects the integrative nature of the SoS and its ecosystem. Our future work will include additional literature searches and interviews with engineers designing SoS for recurring use of specific criteria for evaluating the health of the ecosystems supporting a SoS.

References

Amorim, S. d. S., Almeida, E. S., McGregor, J. D., and Chavez, C. v. F. G. (2014). When ecosystems collide: Making systems of systems work. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW '14, pages 1–4.

Feiler, P. H., Sullivan, K., Wallnau, K. C., Gabriel, R. P., Goodenough, J. B., Linger, R. C., Longstaff, T. A., Kazman, R., Klein, M. H., Northrop, L. M., and Schmidt, D. (2006). *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Software Engineering Institute.

Iansiti, M. and Levien, R. (2000). Keystones and dominators: Framing the operational dynamics of business ecosystems.

Klein, J. and McGregor, J. D. (2013). System-of-systems platform scoping. In *4th International Workshop on Product Line Approaches in Software Engineering (PLEASE).*

Maier, M. (1998). Architecting principles for systems-of-systems. *System Engineering*, 1(4):267–284.

Designing a Software Architecture for a Railway Safety Platform

Adailton Lima[1], Rodrigo Reis[1], Melina Alberio[2], Carlos Lopes[1], Cleidson de Souza[1,2]

[1]Software Engineering Laboratory (LABES) Federal University of Pará (UFPA) Belém, Pará, Brazil

[2]Vale Institute of Technology (ITV) Belém, Pará, Brazil

Abstract. *We were faced with the challenge of designing solutions to increase the safety of all people involved with two corporate railways: employees working on transportation and maintenance services and citizens who live along the railroads. Instead of focusing on different solutions, we want to design a software platform that will allow the evolution of a software ecosystem. In this paper, we present the initial design and assessment of the software architecture of this platform. We briefly report the current architecture itself, focusing on some of the design decisions we made as well as our evaluation of the ATAM method to support the design of software platforms.*

1. Introduction

A large mining company headquartered in South America funded us to develop new information and communication technologies aiming to increase the safety of employees and citizens who work or live close to the companies' railways. In addition to the risks to the company's employees and contractors, there are several cities and communities along these railways. This means that the technologies being designed in this project should not only improve the safety of company's employees, but also minimize the risks to the people who live along the railways.

The authors then faced an interesting challenge: *how to design technology to enhance the safety of company's employees, contractors and citizens who live along these railways?* The typical approach would be to identify the main risk scenarios and then design solutions that were specific to these scenarios. However, instead of doing that, we have decided to build a software platform to provide a broader set of services that can be used by different solutions for railroad safety. We then faced the problem of designing a software platform as a problem of designing a complex software system with a number of important questions related to the design of its underlying software architecture. As such, we chose to adopt a method for designing software architectures; in this case, we adopted the ATAM (Architecture Tradeoff Analysis Method) method [Kazman 2002]. This paper reports the software architecture we designed including the design decisions we have made, as well as an assessment of the ATAM as a method for designing software platforms.

2. Software Ecosystems, Platforms and their Design

According to [Bosch 2009], "a software ecosystem consists of a software platform, a set of internal and external developers and a community of domain experts in service to a community of users that compose relevant solution elements to satisfy their needs". The question then is *how to design such a software platform*? How does the underlying software architecture of this platform should look like? What are the recommendations, guidelines or methodologies that can be used to design such a software platform?

Related work to ours can be grouped in a small number of groups. In the first group, we find papers describing how organizations moved from a software product to a software (platform) ecosystem. Examples of this approach include [Costa et al. 2013] and [Bosch 2009], who discuss how to move from a software product line to a software ecosystem. Another group of related work focuses on modeling software ecosystems as a way to understand how they evolve over time (see for instance, [Monteith 2013]). Finally, the third set of related work focuses on the software architecture of software ecosystems. This includes the analysis of how extensible the APIs of these platforms are and the architectural challenges faced in software ecosystems [Bosch 2009].

Since we were not able to find an approach that could fit our particular context, we decided to adopt the ATAM method to guide the design of our software architecture. We chose ATAM because it is a fairly known method for architectural definition and assessment. In the following section we will very briefly describe our usage of the ATAM method.

3. Using the ATAM Method

As a process of "risk identification", the ATAM can be conducted in an early stage of the project, where there is no detailed information or implementation available but requirements that are used to start a discussion on the decisions about the software architecture. One of the first steps we took was to identify the stakeholders related to the railroad safety platform development. The main stakeholders are:

- **S1**: Company's information technology (IT) group responsible for the deployment of new information and communication technologies;
- **S2**: Researchers developing new information and communication safety solutions that will eventually be integrated into the software platform; and
- **S3**: Internal or external software developers that need to access platform services and data to create new safety solutions.

S1 stakeholders are aware of the current infrastructure constraints and future expansion plans. Their strategic view about the IT infrastructure is essential to understand the limits that new safety services and technologies will face. Their participation is also important to provide an overview of possible new technologies needed to upgrade the IT services of the company. The participation of S2 stakeholders is essential to identify the services, data and quality attributes that the software platform must provide. Finally, S3 are stakeholders responsible for creating new safety solutions that will exist on the platform. These solutions might be *clients* of data/services provided by other solutions or *providers* of data/services to other solutions.

Most of the identified quality attributes are related to the communications infrastructure that allow the services and data providers to use channels to provide online monitoring and notification in the railway. With these quality attributes we designed an early version of the software architecture based on a component and layered view of services, devices and applications involved in the safety platform. This step was very important to create a shared view that included all components working together, since before that, the team only had a vague notion of the required services, without knowing how to integrate them.

Following the method, we interviewed an experienced S1 stakeholder. This stakeholder is a member of a larger team that is defining new communication technologies to be acquired and deployed on one of the company's railways. His participation was important to validate the assumptions and quality attributes that should be provided by the communication services to support the actual railway safety platform. This was critical due to the geographical distance between our team and the IT infrastructure team (other S1 members). To complicate things, this railway crosses the Amazon rain forest, and no detailed information about its infrastructure was available at the beginning of the project. Through the S1 stakeholder, we confirmed the existence of maintenance shafts distributed alongside the railway. However, not all of them have energy and network connectivity to guarantee high availability, one of the initial platform requirements.

As a result of the analysis of the interviews and initial architecture design, we defined different alternative scenarios for the designed architecture. In one of them we moved distributed services to a central server. This led us to an architectural tradeoff, where we must decide between (i) the cost to create the infrastructure to support local servers spread along the railroad and (ii) the cost to communicate messages and notifications in only one central server with a potential bottleneck for system performance. This is only one example of an architectural trade-off that we faced as the result of the ATAM method. Due to space constrains we cannot report all of them.

4. Assessing the ATAM Method

According to [Taylor 2013], a successful software ecosystem is the result of good architectural styles and design decisions in order to create an open environment that supports the success criteria of the platform. This does not mean that a good architectural style is sufficient for a successful software ecosystem, but instead it is an important and necessary condition. In fact, there are social and economical factors that influence the development of software ecosystems [Barbosa 2013].

Before we conducted the design and analysis process, the stakeholders were only aware of their "safety apps", i.e., the software and/or hardware solutions they were implementing. At the time, none of them have thought about how their assumptions were affected by other components or the existing IT infrastructure. As a practical result, S2 and S3 stakeholders are now researching alternative designs to support their solutions because of the likely scenario of lack of communication with local servers.

It is important to mention that the same strength of ATAM is also a weakness. To be more specific, ATAM suggests that workshops and/or interviews should be conducted with important stakeholders, and especially software developers who will implement the architecture and, in our case, create solutions for that platform. However, one of the most important aspects of any software ecosystem is the possibility to tap into the talent of software external developers, i.e., developers who will create solutions that have not been imagined before. This means that the ATAM results can help explore the architectural space of the "known" solutions reported by the interviewed developers as well as solutions that are "similar" to these. If new solutions challenge the assumptions embedded in this architectural space, the architecture of the software platform might fail. Despite that, we still believe that ATAM provided an interesting starting point for us because it guided us through the process of designing the architecture of the software platform by documenting requirements and forcing us to think about design trade-offs.

5. Conclusions

An important aspect of any software ecosystem is its software architecture [Taylor 2013]. This architecture needs to be open and flexible to allow software developers contribute with new and innovative solutions. While the research community recognizes the importance of this software architecture, most previous work focuses on the evolving a software platform from a specific software product [Bosch 2009] [Costa 2013]. In other words, there is limited work suggesting how a software architecture should be designed when creating a new software ecosystem.

In this paper, we describe the usage of the ATAM method [Kazman 2012] to guide the definition and assessment of a software platform that is being built to increase the safety of employees and inhabitants who work and live alongside two major railways in Brazil. We also report on our evaluation of the ATAM method in this context, i.e., the advantages and weaknesses of using ATAM to design a software platform for ecosystems.

ACKNOWLEDGMENTS

We would like to thank the funding from CNPq (process number 310468/2014-0) and Chamada 59/2013 MCTI/CT-Info/CNPq (process number 440880/2013-0).

REFERENCES

Barbosa, O., Santos, R., Alves C., Werner, C., Jansen, S. “A Systematic Mapping Study on Software Ecosystems through a Three-dimensional Perspective”. In: Software Ecosystems: Analyzing and Managing Business Networks in Software Industry. Edward Elgar, Cheltenham, UK. (2013).

Bosch, J. From software product lines to software ecosystems, Proceedings of the 13th International Software Product Line Conference, 2009, San Francisco, California.

Cataldo, M. and Herbsleb. J. D. Architecting in software ecosystems: interface translucence as an enabler for scalable collaboration. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume* (ECSA '10), Carlos E. Cuesta (Ed.). ACM, New York, NY, USA, 65-72.

Costa, G., Silva, F. et al., 2013. From applications to a software ecosystem platform: an exploratory study. In *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems* (MEDES '13). ACM, New York, NY, USA, 9-16.

Kazman, R.; Klein, M.; Clements, P.; Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley. SEI Series in Software Engineering, 2002.

Kazman, R.; Michael Gagliardi, William Wood, Scaling up software architecture analysis, Journal of Systems and Software, Volume 85, Issue 7, July 2012, pp. 1511-1519.

Monteith, J. Y. and McGregor, J. D. Hadoop and its evolving ecosystem. In Proceedings of the Fifth International Workshop on Software Ecosystems, 2013.

Taylor, R. N. The Role of Architectural Styles in Successful Software Ecosystems. In: Proceeding of the 17th International Software Product Line Conference, NY, 2013.

A Summary of Challenges for “MDE as Service”

Fábio P. Basso[1], **Toacy C. Oliveira**[1], **Cláudia M. L. Werner**[1]

[1]Federal University of Rio de Janeiro, COPPE - PESC,
Rio de Janeiro, RJ, Brazil

{fabiopbasso,toacy,werner}@cos.ufrj.br

Abstract. *The introduction of resources for Model Driven Engineering (MDE) in industrial contexts is seen as a business opportunity for some companies and professionals. From an ecosystems perspective, resources are analyzed, acquired from repositories, adapted and integrated in software development environments. This paper summarizes some challenges for “MDE as Service”, which require the introduction of these resources in target contexts, considering a perspective for MDE ecosystems.*

1. Introduction

Model Driven Engineering (MDE) achieved a certain maturity in practice and research, leading Software Engineers to the development of several tools, Domain Specific Languages (DSL) and resources that assist tasks for software development [Mohagheghi et al. 2013]. Aiming at introducing MDE in target contexts, some initiatives for “MDE as Service” [Basso et al. 2013, Monteiro et al. 2014] develop and adapt resources such as DSLs, model transformations, Model Transformation Chains (MTCs), etc. This implies in an effort from professionals to analyze and integrate candidate MDE resources that cooperate in one or more Software Development Process (SDP) adopted by a company.

This scenario can be considered from the perspective of software ecosystems (SECO) [Bosch 2009]. To Jansen et al., a SECO is a unit of business where a common technological platform for services and software allows to connect resources, information and artifacts [Jansen et al. 2009]. Although ecosystems gained attention from research in recent years [Bosch 2009, dos Santos et al. 2013, Fuggetta and Nitto 2014], existing work does not identify issues for the implementation of approaches for MDE as Service. Thus, research gaps must be discussed.

This paper presents some challenges to implement this reuse approach and it is organized as follows: Section 2 contextualizes MDE as Service; Section 3 presents our analysis of the challenges to introduce MDE in target contexts from the MDE ecosystems perspective and conclusion is presented in Section 4.

2. MDE as Service

Approaches for MDE as Service [Basso et al. 2013, Monteiro et al. 2014, Mohagheghi et al. 2013] need to deal with resources for MDE reused in an inter-organizational level (i.e., used by one or more software development companies). Bosch makes a distinction between SECO and regular Software Product Line (SPL) approaches, claiming that when a SPL extends the organizational boundary (i.e., intra-organizational), then a software ecosystem is established to manage inter-organizational

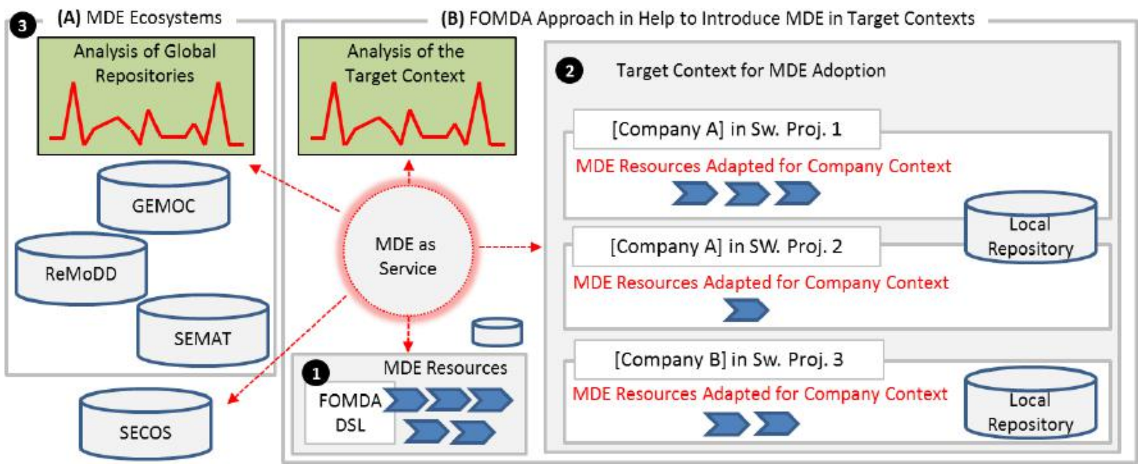

3
(A) MDE Ecosystems
Analysis of Global Repositories
GEMOC
ReMoDD
SEMAT
SECOS
(B) FOMDA Approach in Help to Introduce MDE in Target Contexts
Analysis of the Target Context
MDE as Service
1
MDE Resources
FOMDA DSL
2
Target Context for MDE Adoption
[Company A] in Sw. Proj. 1
MDE Resources Adapted for Company Context
Local Repository
[Company A] in SW. Proj. 2
MDE Resources Adapted for Company Context
[Company B] in Sw. Proj. 3
MDE Resources Adapted for Company Context
Local Repository

Figure 1. A Possible Scenario for MDE as Service.

resources [Bosch 2009]. In [Basso et al. 2013], although not discussing about SECOs, we exemplified a scenario where SPL is applied to manage inter-organizational resources for MDE. Thus, MDE as Service is an interesting scenario where concepts for SECO can be applied.

In this scenario, resources developed for MDE (e.g., model transformations, DSLs and transformation tools) are introduced in different contexts. This is not easy and requires a set of techniques and tool support for reuse that makes the configuration of resources for MDE flexible. An analysis of the target context is carried out, highlighting which resources for MDE are used in the development of a specific software project, e.g., selecting an appropriate DSL to be used in the development of web information systems. It is also important to consider the know how of teams to support the design and development tasks, which may imply on the use of different frameworks, processes and technologies.

In our previous experiences [Basso et al. 2013], the analysis is followed by the adaptation in existing resources for MDE (Figure 1, box 1), delivering at the end a configured tool in conformity with target context (Figure 1, box 2). This is described in the Features-Oriented Model-Driven Architecture (FOMDA) approach [Basso et al. 2013], which includes technical information on how to execute the engineering of adaptive model transformations. The generation of a flexible tool support for MDE is possible through the FOMDA DSL, a language to design MTCs, Feature Model, and to associate features with model transformation components. Thus, resources are customized for intra and inter-organizational contexts.

The current challenge is to support the usage/reuse of resources for MDE developed around the world in a perspective of ecosystems, which makes the FOMDA DSL limited. This is discussed in the next section.

3. MDE Ecosystems

We believe that collaboration is the key to reduce cost in MDE as Service, as illustrated in Figure 1 (A). In other words, instead of developing new DSLs, we can make use of those proposed in the literature of the area, analysing the best options to introduce in inter-organizational contexts. In the following we present the main initiatives for MDE Ecosystems that are inserted in this scenario.

MDE Knowledge Base (KB). ReMoDD [France et al. 2007] is a repository that shares some didactic material for MDE published in some conferences such as MODELS, ECMFA, etc. Most information is available in documents, papers, tutorials, models, metamodels and transformations. In [Mussbacher et al. 2014], the authors claimed that this KB will centralize good practices, but that the lack of critical mass imposes difficulties since we have no habit to share information in the area. ReMoDD, therefore, is a KB in operation that can be important to help in reducing the learning curve.

Globalization of DSLs. In order to share resources for MDE, it is important to ensure that eventual compositions in MTCs are valid. The GMOC initiative is an effort to ensure that technicalities from MDE will be interchangeable in practice [Combemale et al. 2014]. In other words, GMOC will enable a collaborative scenario for MDE considering heterogeneous inter-organizational contexts. Thus, this is important for MDE as Service, since GMOC can help in reducing the costs to introduce MDE in practice.

Knowledge Base for Processes. SEMAT [Johnson et al. 2012] is an initiative to provide a knowledge base in Software Engineering related to process models. This is important because some companies target for MDE adoption have not defined their SDP, making costly the analysis of the target context. SEMAT can help in reducing costs through information about processes. Besides, SEMAT uses Essence as a core representation language, which can be used in the context of MDE to automatically integrate technical resources for MDE with target process models represented with Essence.

Ongoing work. OMG should support a common language for resources associated with these KBs. This language could be helpful to Software Engineers while deciding about a design tool to include in a target software project, making more viable MDE as Service. In order to implement this new scenario, in [Basso et al. 2014], some requirements for this common representation are presented. We proposed RAS++, a DSL that extends the Reusable Asset Specification (RAS) to represent data associated with MDE artifacts. RAS++ aims at facilitating the transition from an information found in a repository (e.g., ReMoDD or GMOC) automatically to target contexts (e.g., representations that integrate these artifacts through MDE Settings such as the FOMDA DSL).

Summary of research gaps. On a perspective of ecosystems, MDE researchers and practitioners could: 1) investigate the applicability of approaches for SECO to promote the reuse of MDE resources, thus helping in the MDE adoption; 2) propose and develop platforms as services for MDE Ecosystems, e.g., finding the requirements for the integration of OSLC [Basso et al. 2014] in this scenario; and 3) propose approaches for a network of collaborative services, connecting people, processes, tools and companies on the support for MDE as Service.

4. Conclusion

MDE as Service is an approach where the introduction of MDE in target software development companies is considered as the core business. In order to reduce costs through shared resources for MDE among organizations, MDE as Service can benefit from approaches that propose the reuse through repositories/Knowledge Bases. This reuse approach, which is related with the ecosystem perspective, is few discussed in the literature of the area related to the MDE specificity. Thus, our contribution is a summary of some research gaps related with promising MDE ecosystems, which can help in future initiatives for MDE as Service.

References

Basso, F. P., Pillat, R. M., Oliveira, T. C., and Becker, L. B. (2013). Supporting large scale model transformation reuse. In *12th International Conference on Generative Programming: Concepts & Experiences.*, GPCE'13, pages 169–178.

Basso, F. P., Werner, C. M. L., and Oliveira, T. C. (2014). Towards facilities to introduce solutions for mde in development environments with reusable assets. In *International Conference on Information Reuse and Integration*, IRI'14, pages 195–202.

Bosch, J. (2009). From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 111–119.

Combemale, B., Deantoni, J., Baudry, B., France, R., Jézéquel, J.-M., and Gray, J. (2014). Globalizing modeling languages. *IEEE Computer, Institute of Electrical and Electronics Engineers*, 47(6):68–71.

dos Santos, R. P., Esteves, M. G. P., de S. Freitas, G., and de Souza, J. M. (2013). Software ecosystems comprehension and evolution. *Social Networking*, 3(2):108–118.

France, R., Bieman, J., and Cheng, B. (2007). Repository for model driven development (remodd). In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4364 LNCS, pages 311–317.

Fuggetta, A. and Nitto, E. D. (2014). Software process. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE '14, pages 1–12.

Jansen, S., Finkelstein, A., and Brinkkemper, S. (2009). A sense of community: A research agenda for software ecosystems. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 187–190.

Johnson, P., Ekstedt, M., and Jacobson, I. (2012). Where's the theory for software engineering? *Software, IEEE*, 29(5):96–96.

Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M. A., Nordmoen, B., and Fritzsche, M. (2013). Where does model-driven engineering help? experiences from three industrial cases. *In Software & Systems Modeling*, 12(3):619–639.

Monteiro, R., Assumpcao Pinel, R., Zimbrao, G., and Moreira de Souza, J. (2014). The mdarte experience: Organizational aspects acquired from a successful partnership between government and academia using model-driven development. In *International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 575–586.

Mussbacher, G., Amyot, D., Breu, R., Bruel, J.-M., Cheng, B. H., Collet, P., Combemale, B., France, R. B., Heldal, R., Hill, J., Kienzle, J., Schöttle, M., Steimann, F., Stikkolorum, D., and Whittle, J. (2014). The relevance of model-driven engineering thirty years from now. In *Model-Driven Engineering Languages and Systems*, pages 183–200.

A Conceptual Map of Model-Driven Development for Systems-of-Systems*

Valdemar Vicente Graciano Neto[1,2], **Milena Guessi**[2,3], **Lucas Bueno Ruas de Oliveira**[2,3], **Flavio Oquendo**[3], **Lina Garcés**[2], **Elisa Yumi Nakagawa**[2]

[1]Instituto de Informática (INF/UFG), Universidade Federal de Goiás, Goiânia, Brazil
[2]ICMC, University of São Paulo, São Carlos, Brazil
[3]IRISA-UMR CNRS/Université de Bretagne Sud, Vannes, France

`valdemarneto@inf.ufg.br, {milena, oliveira, linamgr, elisa}@icmc.usp.br, flavio.oquendo@irisa.fr`

Abstract. *SoS development involves difficult and error prone activities related to the constituents runtime integration, interoperability configuration, and deployment. A Model-Driven Development (MDD) approach for SoS can automate the aforementioned activities. However, there is no consensus about which terminology, tools, or models are more suitable for representing SoS in MDD approaches. This paper brings to light a conceptual map that exposes the main concepts and relations of MDD approaches for developing SoS. We designed our model with on the top of a systematic literature review. The main contribution of this paper is the presentation of the proposed map to the community, sharing the state-of-the-art about MDD for SoS.*

1. Introduction

Systems-of-Systems (SoS) are systems engineered from a set of pre-existing independent systems, so-called *constituents*. Constituents usually are heterogeneous and accomplish missions by means of interoperability [Fitzgerald et al. 2012]. They have a diversity of technologies, communication mechanisms, operation systems, and data representation [DeLaurentis 2007]. In this perspective, software engineering for SoS faces new challenges, such as (i) providing interoperability among constituent systems, (ii) fitting constituents into a cohesive set of to deliver emergent behaviors [Maier 1998], and (iii) correctly deploying SoS [Barbi et al. 2012].

Model-Driven Development (MDD) has been applied to deal with the aforementioned issues imposed by SoS [Farcas et al. 2010]. MDD has been reported in a variety of domains [Farcas et al. 2010]. However, there is still a lack of consensus on the adoption of languages and models for representing constituents with all their particularities, and about the best technologies and practices to use. Then, there is a necessity for a systematization of the relevant knowledge regarding MDD for SoS in order to support decisions about the best MDD practices, tools, and approaches to apply in SoS engineering [Graciano Neto et al. 2014]. Knowledge about MDD for SoS is spread in literature.

Conceptual maps can support SoS engineers in this endeavor. They represent knowledge captured from a diversity of sources, exposing the main concepts of a subject

*This research is supported by FAPEG (grant number 09/2013, ID 2013.009.97100854), and FAPESP (grants 2014/01646-4, 2011/06022-0, and 2013/20317-9).

area and the relations that link them. This paper presents a conceptual map of MDD for SoS. The main contribution of this paper is to propose an artifact that represents the state-of-the-art in MDD for SoS. In particular, the main concepts documented in this model were identified in a Systematic Literature Review (SLR) previously reported [Graciano Neto et al. 2014]. Remainder of this paper is structured as follows. Section 2 describes the background on MDD and SoS. Section 3 presents the conceptual map, explaining the concepts and how they are related to each other, and discusses our findings. Section 4 presents final remarks.

2. Model-Driven Development for Systems-of-Systems

SoS are developed as a modern class of systems to match emerging society's demands. They are formed by pre-existing systems called "constituents", and many of them can participate in an SoS [DeLaurentis 2007]. SoS are engineered to deliver emergent behaviors, i.e., functionalities that can not be individually performed by any of its constituents in isolate which is a result of collaboration among constituents [Fitzgerald et al. 2012]. SoS are classified according to level of managerial independence of constituents, i.e., the autonomy given to a particular system for managing its own resources. They are classified as directed, collaborative, acknowledged, and virtual [Maier 1998]. Distinguishing features are inherent to SoS, such as evolutionary development, dynamic behavior, operational independence, and geographical distribution [Maier 1998]. SoS development is often driven by particular missions (e.g., goal, functionality, or set of tasks) that can only be accomplished by the SoS as a whole [Silva et al. 2014].

MDD has been considered for SoS development due to the facilities it offers. It uses models, metamodels, and model transformations to automatically generate code. A *model* is, essentially, an abstraction of reality represented in a textual or visual language (e.g., a Domain-Specific Language, DSL), and models must conform to their respective metamodels. *Metamodels* are special models that guide and restrict how to construct other models. Models and their respective metamodels are submitted as input to model transformers and, through the use of *model transformations*, they transform the source models into target models or code.

MDD approach contributes to SoS development since it [France and Rumpe 2007]: (i) supports the visualization of the whole SoS, mastering complexity related to its large dimensions issues; (ii) faces problems related to large configuration files used for middleware configuration and constituents deployment, converting this error-prone task into a modeling task (more abstract); (iii) transforms models in correspondent software code and configuration files, automating such task, reducing errors, and maximizing quality, productivity, and traceability; and (iv) supports an adequate deployment, providing also maintainability. For this scope, we use MDD as the acronym to designate all model-driven approaches.

3. A Conceptual Map of MDD for SoS

Conceptual Maps represent knowledge [Novak and Ca?as 2006]. They facilitate the understanding on a topic, in a simple graphical format. They can be used for a diversity of purposes, and have been introduced into work environments for problem solving purposes. Aiming at obtaining knowledge related to the development of SoS based on MDD approaches, we previously performed an SLR [Graciano Neto et al. 2014]. As result, we identified 12 studies related to MDD for SoS. After that, a conceptual map was established according to the following steps:

Step 1: Definition of a *core conceptual map*;

Step 2: Analysis of each study and identification of important concepts. Those concepts were associated to the core conceptual map, creating thus, a conceptual map for each study;

Step 3: Combination of the 12 conceptual maps into a large conceptual map, using the core conceptual map as a reference. This is the main artifact proposed by this research.

A complete version of the conceptual map is externally available[1]. According to our map, a *MDD Approach for SoS* can solve one or more *problems*, offer one or more *advantages*, and it can be applied to one or more *domains*. Such an approach comprises *models*, *metamodels*, *transformations*, and *tools*. A model must necessarily conform to exactly one metamodel, and a model represents one or more *SoS features*. A MDD Approach is supported by adequate *tools*.

According to the complete version of the conceptual map:

An SoS can be classified as Directed, Acknowledged, and Collaborative (Virtual SoS did not appear in the SLR). SoS have received the following denominations in the literature: NetCentric SoS (which requires a Virtual Machine to run), Large-Scale Network-Centric Embedded SoS, Large-Scale Distributed Real-Time Embedded System, Interconnected IT Landscape, and Federation of Constituents.

An MDD Approach for SoS can be considered as a Systems Engineering Approach. MDD Approaches for SoS can provide an important support for realizing tasks such as: composing constituents on COTS with middleware support, and handling text files (configuration and deployment files). Furthermore, MDD can offer manageable approaches for dealing with the diversity of technologies, data representation, operating systems, and languages of constituents; the independent function of constituents; the increasingly size and complexity presented by configuration files that are demanded for configuring and deploying SoS; and considerable complexity of large-scale SoS. MDD Approaches have already been applied for the SoS conception in the following domains: Water Management Policies Systems, Air and Ground Traffic in Airport, Flight Booking, Air Force, Flight Control Systems, and Avionics.

A Model can represent one or more SoS features such as Interoperability, Architecture, Missions, Self-Management, Emergent Behavior, and Constituents. An SoS can execute one or more missions. As observed in the included studies, a mission can be represented by a Mission Scenario (or View), and Colored Petri Nets. A Mission is composed by Mission Parts which are structured in software code (usually in the constituents). A Constituent can be represented as an Agent, and each constituent presents a Behavior. The set of constituents' behaviors can form an Emergent Behavior at SoS level. Self-Management is an SoS feature that can be modeled using SelfMML. Constituents are realized by a Middleware Configuration, Mission Code, and COTS. These COTS can be hardware or software. An Architecture can be expressed as a set of views. A view can be a Deployment view, a Structure View, an Activity View, and a Protocol Definition View.

Transformations required by MDD Approach can be accomplished with tool support. Transformations can be written using a Transformation Language. SoS are modeled using SoS Modeling Language. Examples of SoS Modeling Languages include AADL (Architecture Analysis and Design Language), BPMN, CML (COMPASS Research Group Modeling Language, a formal language), COMPASS (Composable Adaptive Software Systems), DEVSML, MATLAB, OPL and OPD (Object-Process Language and Diagram), SelfMML, SysML (a recurrent language), Simulink, UML, WSDL, and XML. DEVSML is a part of DEVS framework supported by Dunip, which is capable to model simulation environments. As transformation languages, we can highlight oAW, XText, XSL, and XSLT, which are part of EMF (Eclipse Modelling Fra-

[1]Conceptual Map, http://goo.gl/3mW4D5

mework). Tools include ACTUAL [Barbi et al. 2012] (Automation of the Configuration and deploymenT of distribUted AppLications), (a Middleware Platform), CoSMIC, GME/GMF, and INGENME.

Advantages of using MDD for SoS engineering include: Analysis, abstraction of constituents and interfaces, automation, design precision, communication between stakeholders is facilitated, high-configurability, high-confidence code generation, interoperability among models, knowledge capture, maintainability, productivity, raising abstraction level, reuse, reduced development risk, simulation, traceability, validation.

4. Final Remarks

This paper presented a conceptual map covering MDD approaches for SoS, offering a panorama of the area of MDD for SoS. It captures a collection of relevant concepts and relations among them. Such map was conceived as a result of an SLR previously carried out. We expect to contribute to the SoS community by offering an starting point from where new researches could be conducted. Contributions include (i) a list of languages currently used or recommended to model SoS, (ii) a collection of the main denominations SoS have received, (iii) a catalog with the main technologies used to engineer SoS with MDD approaches, (iv) a list of the main problems reported by studies as recurrent for SoS development, (v) the main advantages which motivates the adoption of MDD in an SoS development effort, and (iv) prominent domains where MDD have been successfully applied for SoS engineering. Future works include (i) construction of other conceptual artifacts from this map, (ii) consolidating such a model as representative for the area, and (iii) conceiving metamodels for MDD approaches using this conceptual map as a starting point.

Referências

Barbi, E., Cantone, G., Falessi, D., Morciano, F., Rizzuto, M., Sabbatino, V., and Scarrone, S. (2012). A model-driven approach for configuring and deploying systems of systems. In *SoSE*, Genoa, Italy.

DeLaurentis, D. (2007). System of systems definition and vocabulary. Technical report, School of Aeronautics and Astronautics, Purdue University, West Lafayette.

Farcas, C., Farcas, E., Krueger, I., and Menarini, M. (2010). Addressing the integration challenge for avionics and automotive systems from components to rich services. *Proceedings of the IEEE*, 98(4):562–583.

Fitzgerald, J., Bryans, J., and Payne, R. (2012). A formal model-based approach to engineering systems-of-systems. In Camarinha-Matos, L. M., Xu, L., and Afsarmanesh, H., editors, *Collaborative Networks in the Internet of Services*, volume 380 of *IFIP AICT*, pages 53–62. Springer Berlin Heidelberg.

France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *FOSE 2007*, pages 37–54, Minneapolis, MN, USA.

Graciano Neto, V. V., Guessi, M., Oliveira, L. B. R., Oquendo, F., and Nakagawa, E. Y. (2014). Investigating the model-driven development for systems-of-systems. In *SESoS*, ECSAW '14, pages 22:1–22:8, Vienna, Austria. ACM.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

Novak, J. D. and Ca?as, A. J. (2006). The theory underlying concept maps and how to construct them. Technical report, Technical Report IHMC CmapTools 2006-01.

Silva, E., Cavalcante, E., Batista, T., Oquendo, F., Delicato, F. C., and Pires, P. F. (2014). On the characterization of missions of systems-of-systems. In *Proc. of the ECSAW*, pages 26:1–26:8, Vienna, Austria. ACM.

A Preliminary Study of the Adherence to the Communication Maturity Model in Four Software Organizations

Nelson Leitão Júnior[1], Ivaldir Farias Junior[2], Sabrina Marczak[3], Rodrigo Santos[4]

[1]CESAR.EDU

[2]CIn – Universidade Federal de Pernambuco (UFPE)

[3]FACIN – Pontifícia Universidade Católica de Porto Alegre (PUCRS)

[4]PESC/COPPE – Universidade Federal do Rio de Janeiro (UFRJ)

ngslj@windowslive.com, ihf@cin.ufpe.br, sabrina.marczak@pucrs.br, rps@cos.ufrj.br

Abstract. *This paper aims to classify four software organizations with experience in distributed software development in the Communication Maturity Model (C2M), along with a brief introduction of the model itself and the assessment method used to identify the maturity level in each company.*

1. Introduction

Communication has regularly been reported as one of the main challenges in distributed software development (DSD) for several reasons. For instance, team members work in time zones with no overlapping hours and cannot synchronously meet without one of the parts changing working hours. In other cases, miscommunications take place because team members do not speak the same language or share the same culture. In addition, given the lack of face-to-face opportunities to chat, even with advances in technology, communication frequency is still often low when compared to co-located development. It can cause delays in decision-making or clarification requests, for example, jeopardizing deadlines and work progress.

Given this context, we proposed the C2M, a communication maturity model that aims to help companies to improve communication in their distributed projects [Farias Junior 2014]. However, for organizations to be able to apply the C2M and identify how mature their communication practices are, an assessment method is needed. We briefly introduce our model and the strategy used to assess the maturity of four organizations, which is an initial draft of a to-be-proposed assessment method as well as the results of the assessments in order to share with the community our initial insights on the matter.

2. C2M Model

The most appropriate way for measuring the organizational maturity on a discipline (or domain) seems to be a maturity model approach [Alonso and Soria 2010] and that is the role of C2M in communication discipline. The C2M is based in four existing maturity models: CMMI, eSCM, MR-MPS, and Wave. It specifies four maturity levels: *casual*, *partially managed*, *managed* and *reflective*, each composed of a set of practices. Details can be found in [Farias Junior 2014].

3. C2M Assessment Method: A Draft Proposal and Pilot

Inspired on the MA-MPS assessment method [Softex 2012], we proposed a set formularies and interviews with project stakeholders as a mechanism for data retrieval to assess communication in DSD projects. We piloted our assessment strategy in four organizations: *A* (medium, Americas & Asia, 10 years of experience in DSD), *B* (micro, Americas, 2 years of experience in DSD), *C* (medium, Americas, 9 years of experience in DSD), and *D* (medium, Americas, 11 years of experience in DSD). During an interview, C2M practices used within an organization were identified by verifying evidences they took place (or were absent) and the implementation level of each practice. Results are represented by the means of a characterization method, based on the rules of the MA-MPS [Softex 2012].

The assessment proceeded as follows: *Step 1* – the characterization was applied in extracted data from the assessment of the practices in the organizations that were selected for adherence verification; this task resulted in a data-mapping table; *Step 2* – the data mapping table was aggregated in a format of practices on its respective maturity factors and exposed an aggregation factor which was named as 'adherence'. This factor was calculated according to the following formula: *adherence = (more than 50% of incidence of '**T**otally implemented' or '**L**argely implemented' practices)*; and *Step 3* – all 'adherence' values were aggregated for each C2M maturity level, as a percentage factor that indicates overall adherence on each assessed organization (see Table 1).

Table 1 – Findings

Maturity Level	Maturity factors	Number of practices	Org. A	Org. B	Org. C	Org. D
2	10	25	40%	20%	40%	50%
3	13	24	38%	15%	38%	38%
4	4	9	50%	25%	25%	0%

4. Findings

None of the assessed organizations was completely adherent to any of the C2M Levels, but all of them managed to get some effort in communication practices within their DSD projects. Organization B had the lowest overall score, probably by the fact that is a smaller company, still adjusting its development processes for DSD. In addition, the results indicated that the first C2M level had better adherence scores for organizations A, C and D, which indicates a common acceptance of most communication-related known practices by larger development organizations.

References

Alonso, J. and Soria, I. M. De (2010). Enterprise Collaboration Maturity Model (ECMM): Preliminary Definition and Future Challenges. *Enterprise Interoperability IV: Making the Internet of the Future for the Future of Enterprise*, 9p.

Farias Junior, I. H. De (2014). C2M - A communication maturity model for distributed software development. PhD Thesis. CIN, UFPE, Recife, 287p.

Softex (2012). MPS.BR - Melhoria de Processo do Software Brasileiro, guia geral MPS de software. 58p.

Uma Base de Casos, Problemas e Soluções para Equipes de Desenvolvimento Distribuído de Software

Rodrigo G. C. Rocha[1], Ryan Azevedo[1], Anderson Pinheiro[1], Levy Souza[1], Ivaldir de Farias Junior[3], Gabriel França[1], Silvio Meira[2]

[1]Universidade Federal Rural de Pernambuco (UFRPE - UAG)
55292-270 – Garanhuns – PE – Brasil

[2]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50732-970 – Recife – PE – Brasil

[3]SOFTEX
50030-120 – Recife – PE – Brasil

{rodrigo,ryan,gabriel}@uag.ufrpe.br, {anderson.pinheiro27, levybatera,ivaldirjr}@gmail.com, srlm@cin.ufpe.br

Abstract. *This paper presents the modeling of a case base, problems and solutions, from distributed projects reports. The purpose is to assist and mitigate problems faced by stakeholders through the use of past experiences.*

Resumo. *Neste artigo é apresentado a modelagem de uma base de casos, com problemas e soluções, provenientes de relatos de projetos distribuídos, com intuito de auxiliar e mitigar problemas enfrentados por stakeholders através da utilização de experiências passadas.*

1. Introdução

Para auxiliar a falta de compartilhamento de informações em projetos distribuídos, esse trabalho se baseou no conceito de raciocínio baseado em casos (RBC), que é uma abordagem que utiliza o conhecimento específico de situações problemáticas concretas anteriormente vividas (chamados de casos) para enfrentar os novos [Pantazi, Arocha e Moerh, 2004].

Um caso guarda vários atributos e valores, não é uma regra, é uma experiência passada. Conforme dito por Wangenheim (2003), todos os casos são independentes e seus componentes básicos são: o problema, que aponta as características e restrições da situação onde se encontra o problema; a solução, que representa o modo como o problema foi solucionado; e efetividade, que define a a efetividade Identifica o resultado da aplicação da solução na situação do problema descrita, seja ela de boa efetividade ou não. Não é um componente obrigatório.

Desta maneira, o objetivo deste artigo é propor uma base de casos que foi modelada a partir da descrição de problemas e soluções, para que essa possa auxiliar e mitigar problemas enfrentados por stakeholders através da utilização de experiências passadas.

Para execução deste trabalho, dois métodos de pesquisa foram executados, o primeiro foi um survey, aplicado através de um questionário, com intuito de obter informações de pessoas que fazem parte do mercado de software, nesse caso, o questionário contou com a participação de 21 participantes. O segundo método foi uma revisão de literatura com alguns passos sistemáticos, onde os artigos seguiam alguns critérios para seleção e extração de informação. Dessa forma, um total de 102 casos foram criados.

2. Resultados

Na Tabela 1 são apresentados alguns exemplos de casos retirados da base. Conforme mencionado, um caso é composto do problema, da descrição e opcionalmente da efetividade deste. Nesse caso, o exemplo abaixo contempla apenas os problemas e as soluções. E nesse caso, a primeira coluna diz respeito ao problema e a segunda coluna diz respeito a solução utilizada.

Na primeira linha é possível visualizar um exemplo de um problema que diz respeito a um problema que se dava pelo não cumprimento de deadlines, atraso na maior parte das entregas acordadas entre time de desenvolvimento e equipe de negócio da empresa, E nesse caso, como solução, foi a adoção da ferramenta de gerenciamento de projeto Basecamp. Esses exemplos foram escolhidos de forma aleatória, apenas com objetivo de exemplificar parte dos casos contidos na base.

Tabela 1. Exemplos de casos contidos na Base

Problem	**Solution**
Failure to comply with deadlines, i.e., delays in most deliveries agreed between the development team and the company's business crew.	Adoption of Basecamp project management as the issue tracker tool.
We had problems related to the synchronization of working hours because part of the team works in a different time zone with 4 hours difference.	We define what we call "core hours", which is the period of time of intersection of the entire team. During this period, all team members try to stay dedicated to the project.
The usage of a tool called XPlanner for sofware development based on agile teams (mainly using XP).	The tool itself helped in monitoring but did not contribute to the collaboration of the team, mainly because the tool functionalities were restricted.

A partir dessa base de casos, qualquer organização com projetos distribuídos que esteja trabalhando ou pretendendo trabalhar com sistemas de RBC pode utilizá-la como referência para resolução de problemas. O crescimento dessa base se dá pela simples utilização do sistema, uma vez que assim novos casos são criados. Como trabalhos futuros, os dois métodos de pesquisa podem se repetir, só que dessa vez com maior extensão no sentido de captar mais informações.

Referências

Pantazi, S., Arocha, J., and Moehr, J. (2004). Case-based medical informatics. BMC Medical Informatics and Decision Making.

Wangenheim C. e Wangenheim, A. (2003). Raciocínio Baseado em Casos. Barueri: Editora Manole Ltda.

Research Opportunities for Mobile Software Ecosystems

Awdren Fontão[1], Rodrigo Santos[2], Arilo Claudio Dias-Neto[1]

[1]ExperTS/ICOMP – Federal University of Amazonas (UFAM)
[2]PESC/COPPE – Federal University of Rio de Janeiro (UFRJ)

`{awdren, arilo}@icomp.ufam.edu.br, rps@cos.ufrj.br`

Abstract. *Software solutions are collaboratively built within a dynamic and global market, often requiring adaptation of software development processes. This trend has been broadly studied as Software Ecosystem (SECO); in the mobile platform domain, named Mobile Software Ecosystem (MSECO). In this paper, we pointed out research opportunities extracted from 28 papers. The research opportunities were grouped into three dimensions (Business, Technical and Social) as an answer for the research question "What are the main research opportunities for a Mobile Software Ecosystem?". The contribution is a research agenda to help the Software Engineering community to understand and research on MSECO.*

1. Introduction

A Mobile Software Ecosystem (MSECO) is a set of a collaborative systems, users and developers that creates complex relationships driven by competition and cooperation within niches, similar to biological ecosystems [Lin and Ye 2009]. In this context, we performed a systematic mapping study, published in [Fontão et al. 2015], in order to map existing technical literature regarding MSECO, in which a total of 28 papers were analyzed. As concluded in this study, it is necessary to analyze research opportunities in MSECO and classify it to a better understanding, since they still remain unclear. In this paper, we then answer an additional research question: *"What are the main research opportunities for a Mobile Software Ecosystem?"*. As a result, we identified research opportunities on MSECO grouped into 3 dimensions (*Business* – focuses on the ecosystem management, *Technical* – focuses on the ecosystem platform, and *Social* – focuses on ecosystem stakeholders) suggested by [Santos and Werner 2011].

2. Research Opportunities for Mobile Software Ecosystems

Business: 1) what are the strategies for development and/or identification of niches within the ecosystem based on publishing, advertising and selling of mobile apps (including successful mobile apps)? 2) how does the ecosystem support developers with approaches, methodologies and tools for software engineering activities, e.g., design, development, publishing, follow-up of mobile apps and interactions among MSECO elements in the development process? 3) how should the management activity benefit from the characterization and modeling of interactions among MSECO elements and factors that affect platform adoption by developers and software reuse/opening mechanisms for developers? 4) how does the quality assurance (QA) strategies depend on the orchestration of the ecosystem and on the specific solutions to each MSECO (e.g., practices to help in producing successful software in a complex environment, guidelines to develop mobile apps and a certification process that helps to preserve the ecosystem quality)? 5) how to cope with different types of requirements for mobile apps considering market change, competitors and users, and also how to support packaging requirements (app description, title, keywords, screenshots and marketing artifacts)? 6) how can maturity levels and value creation in ecosystems help to leverage an MSECO?

Technical: 1) how to model an MSECO architecture taking into account extensions and existing conflicts and/or dependencies among them (e.g., registration process, technical support, testing and distributing privileges)? 2) which factors influence and promote the adoption of APIs and design decisions (e.g., restricted access to libraries, and effects on how developers sell, buy and reuse apps)? 3) which factors should be considered when an MSECO is pursuing strategies for taking care of the platform evolution (e.g., business, communication, knowledge management, development/design techniques)? 4) how to ensure that the integration of an extension preserves the quality characteristics of the ecosystem platform? 5) how should the keystone define quality practices and techniques based on user requirements and quality policies? 6) how to construct tools to help developers to migrate their apps to new ecosystems?

Social: 1) how do developers think and feel about their activities within an MSECO, for example, assuming that an improvement on the Developer eXperience (DX) use to have a positive impact on software development and on developer engagement? 2) Regarding DX and UX (User eXperience), which factors influence attraction and retention of community members (users and developers)? 3) how to define reuse strategies for end-users based on analysis of factors that influence the number of new users and the user retention using prediction models? 4) how to communicate requirements and user feedback to developers, and also how could this information influence the development of a mobile app? 5) how to classify different types of ecosystem communities? 6) how can these communities identify strengths and weaknesses of MSECO documentation and then evolve supporting materials?

3. Future Work

As future work, we intend to analyze our results and related work in order to identify similarities and differences between the research opportunities in SECO and MSECO. This analysis is important to make MSECO domain clear.

References

Fontão, A., Santos, R.P. and Dias-Neto, A.C. (2015) "Mobile Software Ecosystem (MSECO): A Systematic Mapping Study". In: Proceedings of the 39th IEEE Annual International Computers, Software & Applications Conference (COMPSAC), Taichung. To appear.

Lin, F.L.F. and Ye, W.Y.W. (2009) "Operating System Battle in the Ecosystem of Smartphone Industry". In: Proceedings of the 9th International Symposium on Information Engineering and Electronic Commerce (IEEC), Ternopil, pp. 617-621.

Santos, R.P. and Werner, C.M.L. (2011) "A Proposal for Software Ecosystems Engineering". In: Proceedings of the Third International Workshop on Software Ecosystem (IWSECO), Brussels, pp. 40-51.

Investigating Issues of Human-Computer Interaction for Systems-of-Systems

Valdemar Vicente Graciano Neto[1,2], **Lina Garcés**[2], **Clodis Boscarioli**[3], **Elisa Yumi Nakagawa**[2]

[1]Universidade Federal de Goiás (UFG) - Goiânia, GO – Brazil

[2]Universidade de São Paulo (USP) – São Carlos, SP – Brazil

[3]Universidade Estadual do Oeste do Paraná (UNIOESTE) – Cascavel, PR – Brazil

valdemarneto@inf.ufg.br, linamgr@icmc.usp.br,

clodis.boscarioli@unioeste.br, elisa@icmc.usp.br

Abstract. *Systems-of-Systems (SoS) are an emerging class of systems. Such systems receive stimulus from humans and from the environment to trigger the accomplishment of complex missions. Interaction among the SoS, humans, and with the environment must be investigated in order to adapt current interaction engineering methods, establishing advancements to encompass a suitable SoS Interaction. This paper presents first insights on interaction engineering for SoS. We outline the topic and briefly discuss how the SoS's inherent characteristics impact on the interaction design for SoS.*

1. Introduction

SoS are a class of software-intensive systems whose constituent parts are, themselves, pre-existing systems called constituents. Remarkable examples include swarm of robots, Smart Systems (such as Smart Cities, and Smart Buildings), Cyber-Physical SoS (CP-SoS), and Ambient Assisted Living (AAL). New modes of interaction emerge as a result of the inherent features delivered by SoS, such as the nature of the constituents, the emergent behavior, the dynamic architecture, the missions accomplishment, and the operational and managerial independence [Maier 1998]. This paper motivates the investigation and brings preliminary insights on how the nature of the SoS influences the Human-Computer Interaction (HCI) design for SoS[1]. The text is structured as follows: Section 2 reports our insights, and Section 3 the final remarks.

2. HCI for SoS

SoS have been considered the new trend of software systems due to their inherent complexity and large dimensions [Santos et al. 2014]. As such, there is a necessity of migrating the interaction design practices and methods to the software engineering of SoS. In particular, SoS' inherent characteristics directly influence how HCI must be designed. Table 1 summarizes the result of our investigation on how UI design activities are influenced by the SoS' particularities according [Maier 1998, Pérez et al. 2013]. Meilich establishes a preliminary discussion on human-SoS interaction, and predicts the paradigms

[1]This work is supported by Brazilian funding agency FAPEG grant 09/2013, ID 2013.009.97100854 and FAPESP grant 2013/20317-9.

SoS feature	**HCI design**
Nature of constituents	Necessity of particular HCI design for each distinct constituent, and different modes of interaction for each type of constituent
Independence of constituents	Necessity for an harmonical confluence of distinct functionalities in a same interface, for constituents' individual behavior, and for the SoS as a whole (such as, mission accomplishment)
Emergent behavior	Constituent's interaction must also fit commands or functionalities for the accomplishment of missions
Evolutionary development	Constituent's interaction also evolve along the time
Geographical distribution	Long distances require for a distributed HCI design; short distances require consistency in interaction modes
Dynamic architecture	Interactive screens which offer management services for the SoS's owner must offer functionalities to deal with the dynamic architecture
Self-management aspects	Runtime update of the SoS configuration and delivering of decision-making functionalities

Table 1. Discussion on impact of SoS' characteristics on HCI design

of human-constituent and constituent-constituent interactions. However, there is no an HCI approach or even a software engineering perspective in that study [Meilich 2007]. Santos et. al presented results that endorse the importance of interaction in SoS domain through the establishing for quality attributes for SoS, in particular, for the crisis/emergent management domain [Santos et al. 2015]. However, strategies to accomplish such quality attributes still must be proposed.

3. Final Remarks

This paper reported preliminary insights regarding Human-Computer Interaction (HCI) for SoS domain. We discussed how the inherent characteristics of SoS impact on the HCI design in SoS engineering. Further research must be conducted. Future works include the investigations of forms to integrate and carry out the classical software engineering (SE) process with the UI design process, and the integration of UI activities in SE for SoS.

References

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

Meilich, A. (2007). Human systems integration - a system of systems engineenng challenge. In *System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on*, pages 1–6.

Pérez, J., Díaz, J., Garbajosa, J., Yagüe, A., Gonzalez, E., and Lopez-Perea, M. (2013). Large-scale smart grids as system of systems. In *SESoS 2013*, pages 38–42, Montpellier, France.

Santos, D. S., Oliveira, B., Guessi, M., Oquendo, F., Delamaro, M., and Nakagawa, E. Y. (2014). Towards the evaluation of system-of-systems software architectures. In *WDES 2014*, pages 53–57, Maceió, Brazil.

Santos, D. S., Oliveira, B. R. N., Duran, A., and Nakagawa, E. Y. (2015). Reporting an experience on the establishment of a quality model for systems-of-systems. In *27th SEKE*, pages 1–6, Pittsburgh, USA.