UNIVERSIDADE FEDERAL DO PAMPA

Bruno Marcelo Soares Ferreira

Investigando a Integração de Ferramentas com OSLC no Contexto do Desenvolvimento de Software

Alegrete
2020

Bruno Marcelo Soares Ferreira

Investigando a Integração de Ferramentas com OSLC no Contexto do Desenvolvimento de Software

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Dr. Fábio Paulo Basso

Coorientador: Prof. Dr. Maicon Bernardino da Silveira

Alegrete
2020

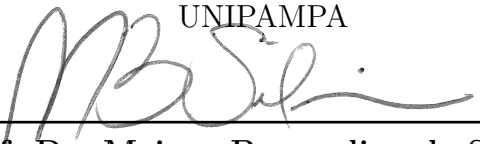
Bruno Marcelo Soares Ferreira

Investigando a Integração de Ferramentas com OSLC no Contexto do Desenvolvimento de Software

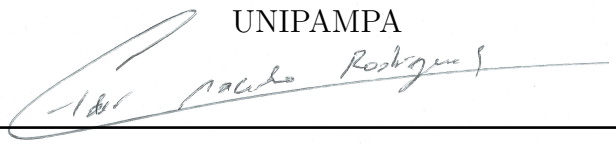
Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 17 de Setembro de 2020
Banca examinadora:

Prof. Dr. Fábio Paulo Basso
Orientador
UNIPAMPA



Prof. Dr. Maicon Bernardino da Silveira
Coorientador
UNIPAMPA



Prof. Dr. Elder de Macedo Rodrigues
UNIPAMPA

Prof. Dr. Rafael Zancan Frantz
UNIJUI

Dedico este trabalho aos meus pais, que sempre me apoiaram
e contribuíram com essa conquista.

RESUMO

A indústria de software investe continuamente em ferramentas modernas para apoiar o ciclo de vida de desenvolvimento de software. Por um lado, esse esforço contínuo na modernização do ambiente de produção permite a introdução de novidades para aumento da produtividade e da qualidade de software. No entanto, também apresenta desafios para engenheiros de software na obtenção de um ambiente integrado de ponta a ponta, como lidar com várias configurações de ferramentas no gerenciamento do ciclo de vida de aplicações. Para mitigar esses desafios, muitas abordagens foram propostas para a integração de ferramentas. Nesse contexto, o padrão industrial e aberto denominado *Open Services for Lifecycle Collaboration (OSLC)* foi proposto para promover a interoperabilidade de ferramentas, principalmente aquelas dedicadas aos ciclos de vida de desenvolvimento de software. Em outras palavras, o padrão OSLC permite a federação de dados ao longo de todo o ciclo de vida de aplicações de Engenharia de Software (ES). O OSLC define uma forma comum de representação dos artefatos criados durante o projeto, bem como funções que permitem a manipulação por outras ferramentas. Este estudo apresenta uma investigação sobre soluções de integração de ferramentas construídas no padrão OSLC, explorando a geração automática de código-fonte de adaptadores OSLC a partir de abordagens de *Model-Driven Engineering (MDE)*.

Palavras-chave: Integração de Ferramentas, Interoperabilidade de Ferramentas, Application Lifecycle Management, Open Services for Lifecycle Collaboration, Engenharia Dirigida por Modelos

ABSTRACT

The software industry continuously invests in modern tools to support the software development lifecycle. On the one hand, this continuous effort towards production environment modernization allows the introduction of novelties to increase productivity and software quality. However, it also presents challenges for software engineers to achieve an integrated end-to-end environment, such as dealing with various tool configurations while managing the application lifecycle. To mitigate these challenges, many approaches have been proposed for tool integration in the context of Software Engineering (SE) enviroments. In this context, the industrial and open standard called Open Services for Lifecycle Collaboration (OSLC) was proposed to promote the interoperability of tools, including those devoted for software development lifecycles. In other words, the OSLC standard allows for federation of data over the entire Software Engineering (ES) application lifecycle. OSLC defines a common interchange format for artifacts created along the software project, as well as allow their manipulation by other tools through functions. This study presents an investigation about tool integration solutions built on the OSLC standard, exploring automatic source code generation of OSLC adapters from Model-Driven Engineering (MDE) approaches.

Key-words: Tool Integration, Tool Interoperability, Application Lifecycle Management, Open Services for Lifecycle Collaboration, Model-Driven Engineering

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ALM *Application Lifecycle Management*

AMS *Asset Management Specification*

BPMN *Business Process Model and Notation*

DSL *Domain-Specific Language*

DTIC Diretoria de Tecnologia da Informação e Comunicação

EMF *Eclipse Modeling Framework*

ES Engenharia de Software

HTTP *Hypertext Transfer Protocol*

IDE Ambiente de Desenvolvimento Integrado

JSP *JavaServer Pages*

MBSE *Model-based Systems Engineering*

MDD *Model-Driven Development*

MDE *Model-Driven Engineering*

OSLC *Open Services for Lifecycle Collaboration*

RDF *Resource Description Framework*

RUP *Rational Unified Process*

SMS *Systematic Mapping Study*

SoaML *Service oriented architecture Modeling Language*

SysML *Systems Modeling Language*

UML *Unified Modeling Language*

URI *Uniform Resource Identifier*

XML *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Ferramentas de software são utilizadas para apoiar profissionais na execução de atividades ao longo do ciclo de vida do software. Essas ferramentas possuem diferentes funcionalidades, configurações, fabricantes e são desenvolvidas sem o suporte nativo para serem integradas com outras ferramentas. Além disso, manipulam artefatos de software (requisitos de software, modelos, código-fonte, casos de teste, etc.) produzidos em diferentes fases do desenvolvimento. Este cenário contribui para que ambientes integrados de ponta a ponta de forma totalmente automatizada sejam raros na indústria (WICKS; DEWAR, 2007).

Organizações de software obtêm novas ferramentas ou desenvolvem suas soluções de software ao longo dos anos, resultando em um ecossistema de software heterogêneo (MESSERSCHMITT, 2005). Isso demanda que as organizações adotem abordagens para melhoria dos processos de software e suporte ferramental que permitam a automação do processo de desenvolvimento por meio da integração de ferramentas.

Entre as alternativas está o *Application Lifecycle Management* (ALM), o qual adota a ideia de integração ponta a ponta, evitando silos de informações devido a características como rastreabilidade, visibilidade e automação de processos (SCHWABER et al., 2006). Para reduzir os problemas relacionados à integração de artefatos de software ao longo de várias fases do ALM, padrões de representação para modelos, metamodelos e transformações têm sido propostas na literatura (KLEPPE; WARMER; BAST, 2003).

Além disso, para mitigar a complexidade desse ambientes, várias ferramentas que auxiliam nas tarefas de Engenharia de Software (ES) foram propostas (EBERT, 2013). Essas ferramentas ajudam os desenvolvedores a aproveitar experiências anteriores ao especificar um novo software, enquanto ferramentas baseadas em modelos ajudam a capturar, organizar e armazenar a maioria das informações trocadas como ativos de software reutilizáveis.

Uma vez que tais ambientes de produção de software são organizados em uma sequência lógica de atividades sob a forma de um ou mais ciclos de vida de gerenciamento de aplicações, o próximo desafio do engenheiro de software é integrá-las em um ambiente de suporte à tarefas. Assim, espera-se automatizar estes ciclos de vida por meio de soluções de integração de ferramentas.

1.1 Caracterização do Problema de Pesquisa

Na Engenharia de Software (ES), o principal desafio da integração de ferramentas é definir como as ferramentas com diferentes características podem trabalhar juntas. Para atingir um objetivo comum, as ferramentas podem precisar usar o mesmo tipo de dados, convenções de interface do usuário e funcionalidades (THOMAS; NEJMEH, 1992). Com relação à integração de dados, há uma gama de diferentes abordagens, opções e desafios para criar, gerenciar, armazenar, reutilizar e acessar dados em aplicativos (WASSERMAN,

1990).

Uma alternativa para integrar dados em ferramentas heterogêneas é transformar, sincronizar e atualizar manualmente os artefatos de ES. No entanto, o paradigma do *Application Lifecycle Management (ALM)* tem uma abordagem diferente, permitindo um fluxo automatizado de compartilhamento de dados gerado durante o ciclo de vida de desenvolvimento do software. O ALM é um paradigma que integra e gerencia atividades de governança, desenvolvimento e manutenção, reunindo desenvolvedores, partes interessadas e usuários finais em um ambiente complexo que promove a colaboração em todos os domínios do projeto. O ALM também inclui o encadeamento de várias ferramentas diferentes, como processadores de texto, compiladores, sistemas de gerenciamento de controle de versão e sistemas de controle de tarefas, todas consumindo e produzindo informações armazenadas como artefatos de software.

O ALM é essencial em abordagens modernas para integrações de *Software Engineering Continuous (CSE)*. Essas abordagens são geralmente implementadas por meio da topologia ponto a ponto (FITZGERALD; STOL, 2017), em uma conexão direta entre duas ferramentas (HOHPE, 2003). Em outras palavras, as ferramentas usadas no ciclo de vida da Engenharia de Software geralmente estão vinculadas a interfaces específicas. No passado, esse acoplamento de ferramentas era uma necessidade para operacionalizar a automação de processos por meio de *Process-centered Software Engineering Environments (PSEEs)* (GARG; JAZAYERI, 1995). Porém, como tais integrações eram caracterizadas por ferramentas fortemente acopladas, quando uma ferramenta é descontinuada e/ou simplesmente substituída no processo, implica em retrabalho e perda de rastros no fluxo de informações ao longo do ALM. Assim, uma vez que o alto acoplamento é uma das razões pelas quais os PSEEs são inviáveis para adoção pela indústria de software (MATINNEJAD; RAMSIN, 2012), alternativas para baixo acoplamento de ferramentas são necessárias.

Em resposta a esta limitação, levando em consideração os cenários de integração do ALM, alguns trabalhos, como (BIEHL et al., 2014), propõem o baixo acoplamento através do provisionamento de ativos de software como serviços. O objetivo principal é minimizar problemas relacionados ao processo de integração de ferramentas adotado em contextos de ES por meio de uma arquitetura típica de serviços (ZHANG; MOLLER-PEDERSEN, 2014). Assim, para suprir as deficiências das soluções existentes, foi proposto o *Open Services for Lifecycle Collaboration (OSLC)* (OSLC, 2020a): uma especificação industrial, como um padrão aberto para interoperabilidade de ferramentas, permitindo federação de dados ao longo da execução de um projeto de software. O *Open Services for Lifecycle Collaboration* (OSLC) define um modelo de representação padrão para artefatos, bem como métodos que permitem a troca de dados entre ferramentas.

Não existe um estudo que mapeie o estado da arte e suas abordagens práticas do OSLC. Como consequência, os engenheiros de software enfrentam dificuldades de

tomada de decisão ao selecionar abordagens para a prática de integração OSLC. Essas dificuldades incluem a falta de dados resumidos para a análise dos requisitos de integração do escopo dos cenários do ALM, bem como dados destacando os benefícios e desvantagens da integração da ferramenta OSLC. Do ponto de vista do pesquisador, também é difícil descobrir as contribuições existentes na área para decidir se devem desenvolver uma nova contribuição/incrementos no estado da arte. Uma terceira lacuna na pesquisa é a falta de um estudo replicável, de forma que engenheiros de software possam replicar estudos ao longo dos anos para acompanhar a evolução da área de pesquisa. Por fim, é escassa a documentação sobre OSLC e suas ferramentas de suporte, o que causa uma longa curva de aprendizado na implementação de soluções de integração por parte dos engenheiros de software.

1.2 Motivação

Para ajudar na integração de ferramentas usadas na produção de software, muitas abordagens foram propostas, interoperando dados entre serviços (BIEHL et al., 2014). As abordagens foco destes trabalhos caracterizam-se por minimizar os problemas do processo de integração relacionados ao desenvolvimento de soluções de integração *ad-hoc* em ambientes complexos. Elas são adotadas nos contextos de ES, tipicamente implementadas por meio de uma arquitetura comum para serviços (ZHANG; MøLLER-PEDERSEN, 2014), o que demanda um protocolo de comunicação comum entre diferentes ferramentas.

Nesse contexto, uma emergente especificação industrial caracterizada como arquitetura comum para ferramentas de Engenharia de Software foi proposta: OSLC. O OSLC é uma padrão aberto para interoperabilidade de ferramentas de software, o qual define um modelo de representação comum para os artefatos produzidos por meio dos domínios do projeto, bem como funções que permitem ferramentas compartilhar dados entre si.

No entanto, estabelecer cadeias de ferramentas OSLC exige dos engenheiros de software o conhecimento de como criar interfaces de ferramentas para a troca de dados interoperada por meio de adaptadores. Essa tarefa não é fácil e requer etapas bem definidas que ajudam os engenheiros de software na automação do processo de desenvolvimento de software.

Com base nisso, este trabalho apresenta uma pesquisa de caráter exploratório, identificando abordagens para o desenvolvimento de soluções de integração, em especial as de *Model-Driven Development* (MDD) e *Model-Driven Engineering* (MDE). Essas abordagens permitem a geração de código-fonte de soluções de integração por meio de *Domain-Specific Language* (DSL), o que pode ajudar na assimilação da teoria da integração e prática (FOWLER; PARSONS, 2011).

1.3 Objetivos

O objetivo geral deste trabalho é explorar o padrão OSLC e suas especificações, bem como as tecnologias associadas com base em MDE para o desenvolvimento de interfaces de ferramentas para a troca de dados interoperada por meio de adaptadores.

Os objetivos específicos são:

1. Identificar o estado da arte e prática do uso do OSLC por meio da execução de um estudo de mapeamento sistemático.

2. Aprender sobre as tecnologias desenvolvidas para suporte ao OSLC. A automação de processos da Engenharia de Software no contexto de desenvolvimento de software, assim como motivado por (FUGGETTA; NITTO, 2014), precisa da investigação exploratória de suporte ferramental para a integração de ferramentas. Assim, pretende-se investigar na prática o padrão OSLC e ferramentas que permitam o desenvolvimento de soluções de integração neste padrão.

3. Aprender sobre as tecnologias de MDE disponíveis para integração de ferramentas por meio do estudo sobre a modelagem e a geração de código-fonte de adaptadores OSLC.

1.4 Classificação de Pesquisa

A metodologia de pesquisa deste trabalho é apresentada com base em quatro categorias propostas por Prodanov (PRODANOV, 2013): (i) natureza da pesquisa, (ii) objetivos da pesquisa, (iii) procedimentos e (iv) técnicas e abordagem do problema.

Este trabalho possui atividades que incluem aprender conceitos e tecnologias para solucionarem problemas específicos sobre a integração de ferramentas por meio do padrão OSLC. Isto caracteriza o estudo como natureza aplicada, pois sugere adquirir conhecimentos para aplicar em uma solução de problema.

Além disso, o objetivo deste estudo pode ser definido como de caráter exploratório, pois pretende-se identificar as características sobre uma determinada área, nesse caso específico a integração de ferramentas utilizando as especificações OSLC.

Os procedimentos e técnicas utilizadas categorizam-se como pesquisa bibliográfica, devido a execução a busca pelos trabalhos relacionados por meio de um *Systematic Mapping Study* (SMS). Um SMS busca identificar o estado da arte e da prática de um tópico de pesquisa de forma abrangente.

Por fim, a respeito da abordagem do problema, esta pode ser classificado como qualitativa e quantitativa. A análise dos resultados dos trabalhos relacionados possui dados quantitativos e qualitativos. Além disso, os dados o relato do uso de uma ferramenta encontrada na literatura para a geração semi-automática de código-fonte também é classificada como qualitativa.

1.5 Contribuições

As contribuições deste trabalho podem ser resumidas nas seguintes atividades:

- Uma revisão da área de pesquisa sobre OSLC por meio de um estudo de um SMS.
- O desenvolvimento de uma solução de integração de ferramentas baseada em OSLC, que incluem as ferramentas Excel/Libreoffice, Redmine e Testlink.
- Um tutorial para o desenvolvimento de adaptadores OSLC utilizando a ferramenta Eclipse Lyo.

1.6 Organização

O presente estudo está dividido em seis capítulos como são apresentados a seguir:

Capítulo 1. Compreende essa introdução. A Seção 1.1 caracteriza o problema de pesquisa. A Seção 1.2 motiva o desenvolvimento do trabalho. A Seção 1.3 aborda o objetivo geral e os objetivos específicos e na Seção 1.4 aborda a classificação da pesquisa. Na Seção 1.5 são apresentadas as principais contribuições deste trabalho.

Capítulo 2. Descreve a metodologia utilizada durante o desenvolvimento deste trabalho. Ou seja, apresenta o desenho da pesquisa adotado para o desenvolvimento do trabalho de conclusão de Curso 1, na Seção 2.1, e o seguimento no TCC 2 na Seção 2.2. As lições do capítulo são apresentadas na Seção 2.3.

Capítulo 3. Apresenta os conceitos e termos relacionados à pesquisa desenvolvida. A Seção 3.1 apresenta os conceitos sobre integração de ferramentas de software. A Seção 3.2 aborda as características do paradigma ALM e a Seção 3.3 descreve o padrão OSLC. A Seção 3.4 introduz os conceitos sobre DSL. A Seção 3.5 caracteriza o estado da arte em OSLC e a Seção 3.6 aborda os conceitos que motivam o SMS. A Seção 3.7 contém um breve resumo do capítulo.

Capítulo 4. Apresenta a busca de trabalhos relacionados por meio da execução de um estudo de mapeamento sistemático. Na Seção 4.1, o escopo o protocolo do estudo de mapeamento sistemático são apresentados. Na Seção 4.2, as questões de pesquisas são respondidas com base nos estudos selecionados e a Seção 4.3 contém a análise dos resultados do SMS. Por fim, na Seção 4.4 são discutidas as ameaças à validade do SMS e as lições do capítulo são apresentadas na Seção 4.5.

Capítulo 5. Aborda a proposta deste trabalho. A Seção 5.1 caracteriza o cenário em que este estudo foi baseado. A Seção 5.2 apresenta a ferramenta escolhida para o desenvolvimento da proposta. A Seção 5.3 apresenta o processo executado para o desenvolvimento de adaptadores OSLC. A Seção 5.4 aborda a demonstração conceitual do desenvolvimento de adaptadores OSLC. A Seção 5.5 apresenta o resumo do capítulo.

Capítulo 6. Aborda as considerações finais sobre os resultados da pesquisa obtidos. A Seção 6.1 discute as limitações da realização deste trabalho além dos trabalhos futuros e a Seção 6.2 contém as produções científicas derivadas deste trabalho.

2 METODOLOGIA

Como um detalhamento da metodologia de pesquisa, este capítulo apresenta um relato sobre as atividades propostas para execução no Trabalho de Conclusão de Curso 1 e 2. Também discute os aprendizados acerca da expectativa inicial de execução de atividades e o trabalho realizado, que precisou de adaptação em decorrência da pandemia de COVID-19. O capítulo é estruturado em uma seção que trata das atividades planejadas e executadas no TCC 1, das adaptações no TCC 2, e de lições aprendidas.

2.1 Desenho de Pesquisa para o Trabalho de Conclusão de Curso 1

A Figura 1 apresenta dois processos elaborados para caracterizar o desenho da pesquisa para o desenvolvimento do trabalho de conclusão de curso 1 e 2. O primeiro processo apresenta os seguintes subprocessos: Concepção, Fundamentação Teórica, Seleção de Alternativas de Implementação e Divulgação. O segundo processo tem duas colaborações de refinamento para atividades envolvendo a fundamentação teórica e implementação, sendo estruturado da seguinte forma: Refinamento do SMS e Alinhamento de Contribuições, Refinamento do Estudo de Avaliação e Testes de Ambientes, Desenvolvimento da Solução de Integração e Divulgação dos Resultados.

A primeira atividade do TCC 1 foi a definição do escopo de pesquisa. A Figura 2 detalha o subprocesso para a concepção do tema de pesquisa. Foram discutidos com

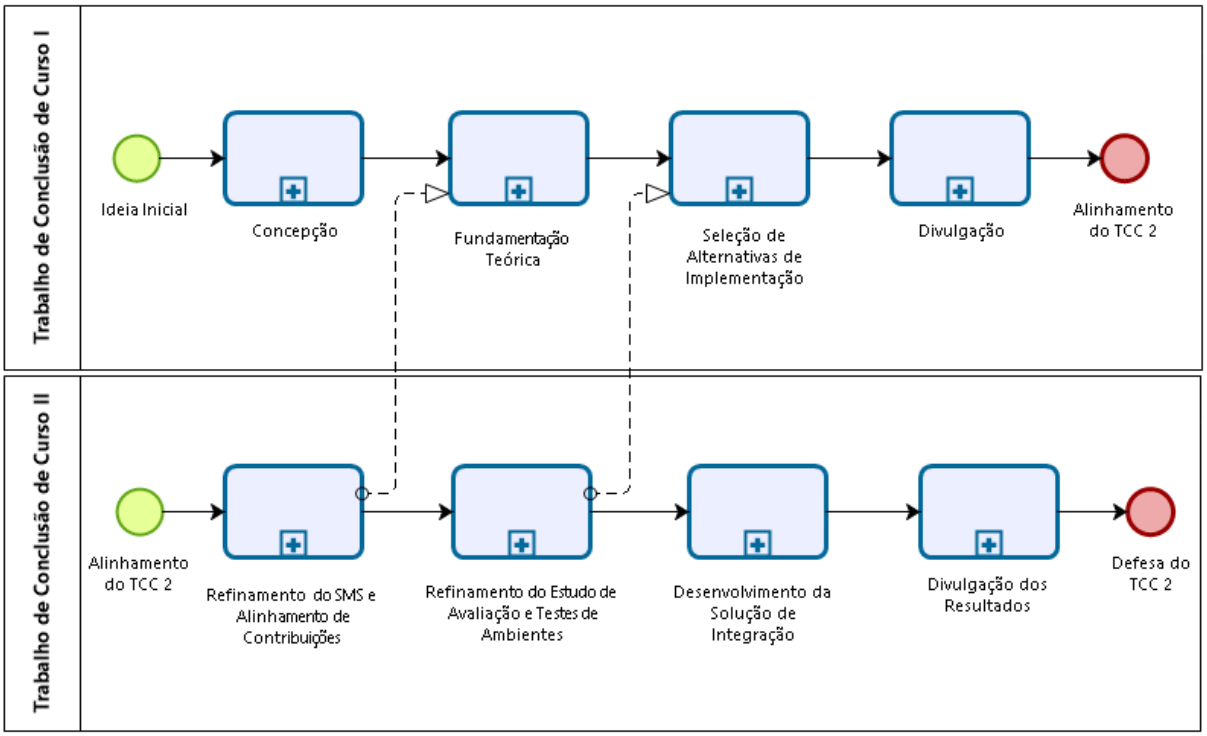


Figura 1 – Processos como Desenho de Pesquisa para a Execução de Atividades do TCC 1 e TCC 2

Fonte: O autor.

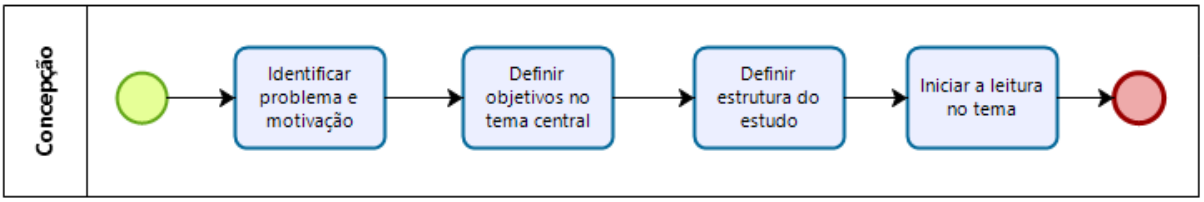


Figura 2 – Subprocesso para a Concepção do Tema de Pesquisa

Fonte: O autor.

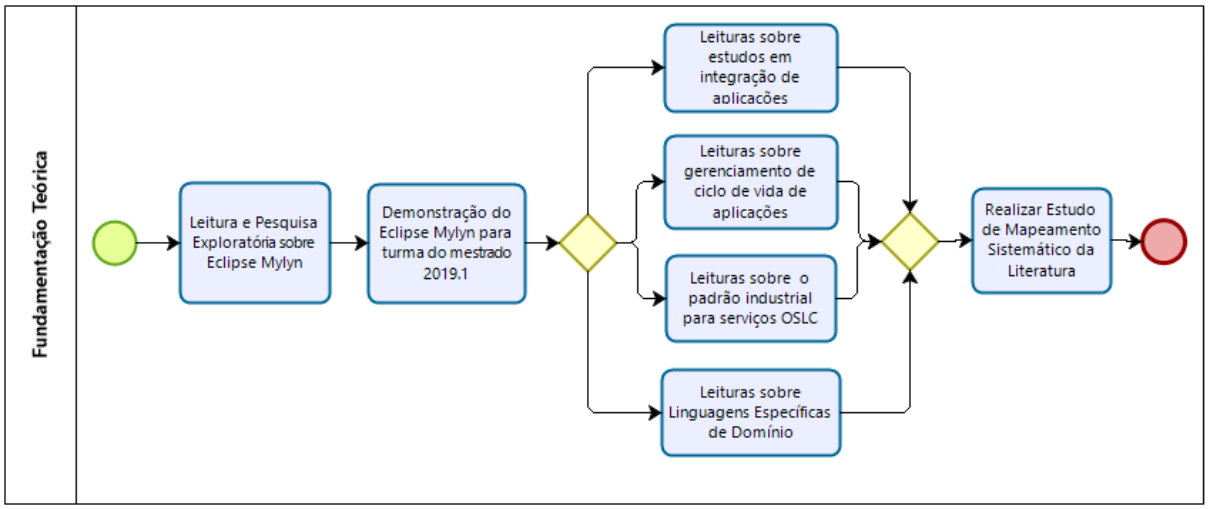


Figura 3 – Subprocesso para a Fundamentação Teórica

Fonte: O autor.

o orientador temas relacionados a integração de ferramentas e os problemas existentes nessas abordagens.

A Figura 3 mostra o subprocesso para a execução das atividades de fundamentação teórica. O início das atividades acontece com a pesquisa sobre a ferramenta Eclipse Mylyn. Esta ferramenta possibilita gerenciar tarefas no ambiente eclipse, contribuindo para a melhor produtividade dos desenvolvedores. Trata-se de uma ferramenta que permite a integração com outros sistemas, como BugZilla, e portanto no contexto de ferramentas utilizadas para ALM.

Após um demonstração conceitual para a turma de mestrado 2019.1, foi discutida a possibilidade de integração com outras ferramentas. Logo, surge como tema de pesquisa uma especificação emergente (OSLC) para integração de ferramentas com o objetivo de automatizar o processo de desenvolvimento de software. Entretanto, não foram encontrados na literatura trabalhos que abordassem o estado da arte e prática do uso do OSLC. Então, a próxima atividade foi realizar um estudo de mapeamento sistemático (SMS).

A Figura 4 detalha o subprocesso para a seleção de alternativas de implementação. Após a execução do SMS, foi possível mapear algumas abordagens para estabelecer soluções de integração com o OSLC. Entre as abordagens está o Lyo, um projeto desenvolvido no ambiente Eclipse que conta com uma DSL para o desenvolvimento de adaptadores

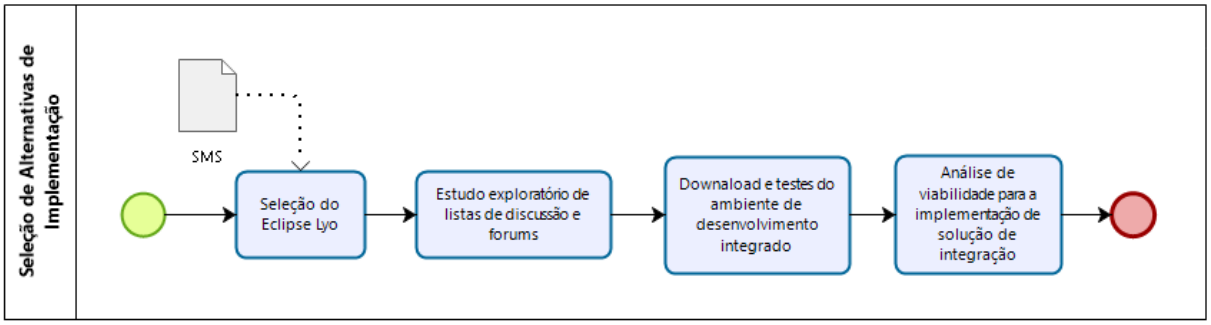


Figura 4 – Subprocesso para a Seleção de Alternativas de Implementação

Fonte: O autor.

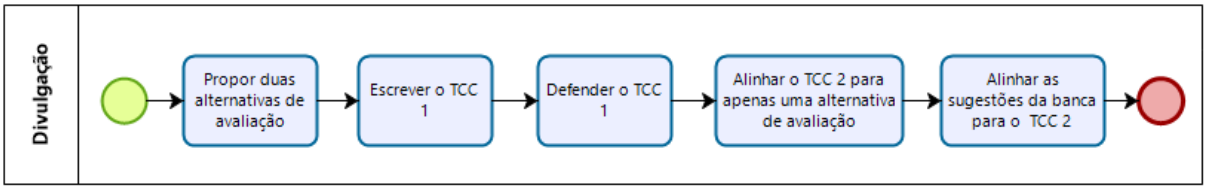


Figura 5 – Subprocesso para a Divulgação do TCC 1

Fonte: O autor.

OSLC. Seu funcionamento foi explorado a partir de listas de discussão em fóruns e projetos disponibilizados na web. Por fim, contatou-se que sua escolha era viável por possibilitar a execução deste trabalho e atingir seus objetivos.

A Figura 5 mostra o subprocesso para a divulgação do TCC 1. Para a defesa do TCC 1, foram propostas duas alternativas de cenários par a avaliação do trabalho, uma para execução em um contexto governamental e ou em um contexto privado. A banca direcionou para a seleção do contexto organizacional governamental, ou seja, o contexto do DTIC. Além disso, foram realizadas as atividades de escrita e a defesa do trabalho. Por fim, após *feedback* da banca, foi possível melhor alinhar o tema de pesquisa para a condução de atividades de implementação e refinamento do relatório de TCC 2.

2.2 Desenho de Pesquisa para o Trabalho de Conclusão de Curso 2

A primeira atividade prevista para execução do processo do trabalho de conclusão de curso 2 trata-se do subprocesso mostrado na Figura 6. O objetivo final era o de relatar o mapeamento sistemático para um artigo de revista da área. Uma vez que se detectou algumas inconsistências na classificação dos estudos para as disciplinas de Engenharia de Software do Rational Unified Process (RUP), foi necessário refinar esta análise com a participação de um terceiro integrante (o orientador).

Estas atividades foram cumpridas na íntegra, resultando no artigo que consta no Apêndice A e superando as expectativas iniciais para fundamentação teórica. Um grande esforço foi necessário para levantar detalhes dos trabalhos selecionados, discutindo

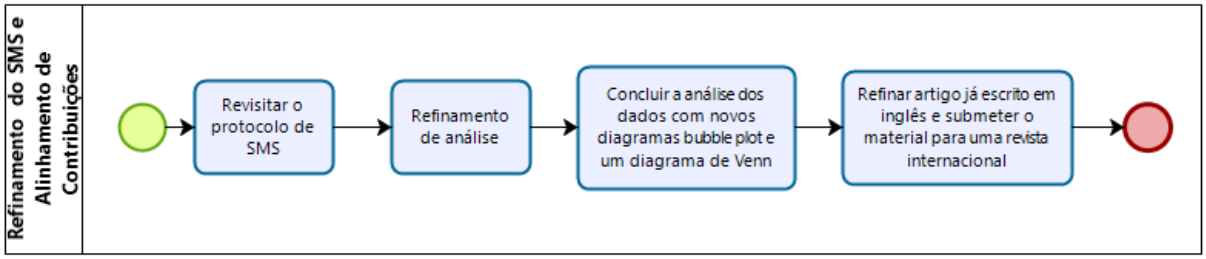


Figura 6 – Subprocesso para Refinamento do SMS e Alinhamento de Contribuições

Fonte: O autor.

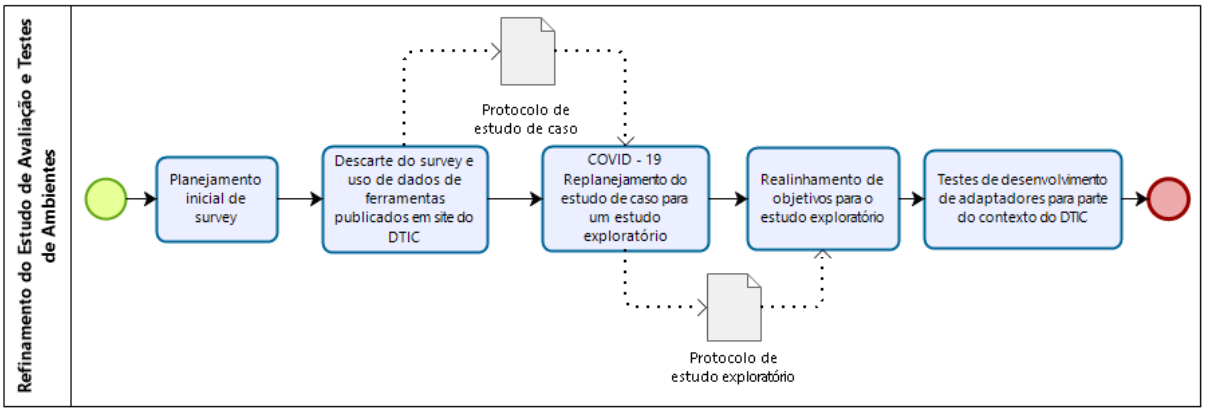


Figura 7 – Subprocesso para Refinamento do Estudo de Avaliação e Testes de Ambientes

Fonte: O autor.

as contribuições em um nível de detalhe que não se encontra nas publicações da área. Portanto, como resultado esperado, existe uma expectativa de envio do material para uma boa revista da área.

No entanto, a execução do subprocesso mostrado na Figura 6 consumiu muito do tempo disponível para a execução do Trabalho de Conclusão de Curso 2. Assim, foi necessário reagendar algumas prioridades, diminuindo o escopo do projeto de pesquisa, focando em um estudo que vai em profundidade no tema na revisão de literatura, mas que deixa a desejar na questão da avaliação empírica, como discutido nas atividades do subprocesso mostrado na Figura 7.

O seguinte subprocesso previa o alinhamento de um estudo de caso em ambiente real, com um time de uma organização governamental DTIC: 1) Concluir plano de avaliação da implantação de uma integração de ferramentas com OSLC no DTIC; 2) Reler os artigos de estudo de caso levantados no SMS; 3) Elencar os atributos de qualidade de propostas para integração de aplicações em OSLC, que iriam direcionar a execução de um estudo de caso proposto para o TCC 2; e 4) Refinar o protocolo do estudo de caso relatado no TCC 1.

A intenção inicial era realizar um estudo aplicado em um time de desenvolvimento, como previsto em protocolo de estudos contido no Apêndice C. As atividades deste pro-

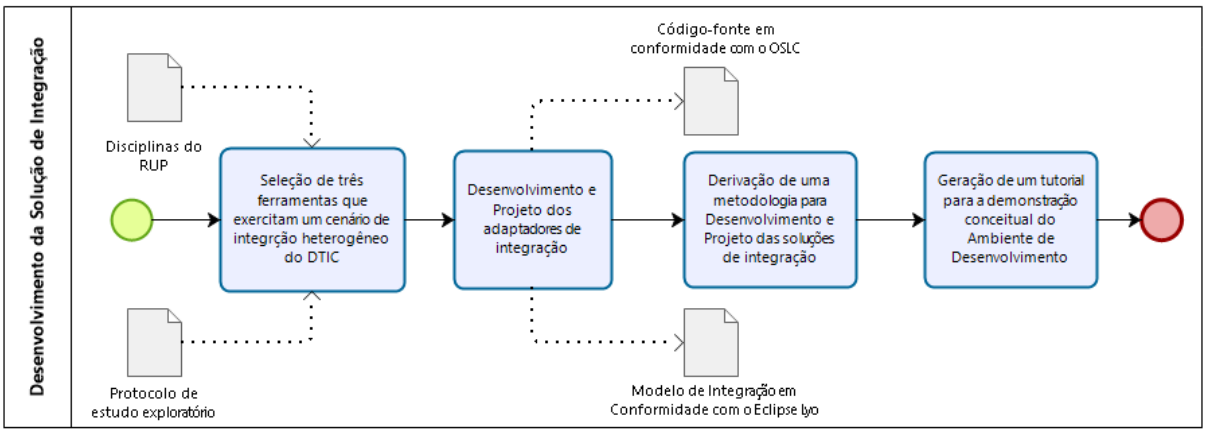


Figura 8 – Subprocesso para Desenvolvimento da Solução de Integração

Fonte: O autor.

tocolo de estudo de caso precisaram ser repensadas devido à pandemia de Covid-19. O time do DTIC se dissipou e se acordou com os pesquisadores da Engenharia de Software que era preferível a execução de estudos que não dependessem de times. De modo à adaptar uma nova avaliação para este cenário, um outro protocolo de estudo de avaliação foi elaborado, seguindo as atividades destacadas no subprocesso "Refinamento do Estudo de Avaliação e Testes de Ambientes".

Na sequência do planejamento para o TCC 2, outra atividade previa a utilização de um suporte ferramental que permite a geração automática de código-fonte a partir de modelos. Portanto, um trabalho aplicado ao tema de *Model-Driven Engineering* (MDE) (BASSO, 2017). Assim, a previsão de atividade foi alinhada no Trabalho de Conclusão de Curso 1, no cronograma definida pelas seguintes atividades: 1) Desenvolver os adaptadores de toolchain para as ferramentas identificadas no *survey*, colocando em operação uma solução de integração para o DTIC; 2) Desenvolver/representar os modelos de integração ou no Eclipse Lyo ou no Guaraná DSL; 3) Desenvolver os geradores de código-fonte para a geração dos adaptadores de ferramenta na plataformas de integração OSLC estudada; 4) Gerar e testar a solução de integração para o cenário

Não foi necessária a execução de um *survey* para formalizar as ferramentas usadas no DTIC, porque encontrou-se uma página que já caracterizava isso. Logo, a 1 foi descartada. A atividade 3 não foi desenvolvida porque identificou-se que o Eclipse Lyo não possui uma sintaxe expressiva que leve à geração de código completo do adaptador. Assim, considerou-se que era preciso encontrar uma DSL mais representativa, o que requer um tempo maior para a execução de outro estudo exploratório do que o previsto no cronograma. Portanto, a atividade 3 foi descartada. Já a 5 não pode ser executada também devido à pandemia, procurando assim a realização de um estudo exploratório com força empírica menor.

Figura 8 mostra o subprocesso que caracteriza as adaptações para as atividades previstas no TCC 1. Ela demonstra que, apesar das readequações, cumpriu-se com o

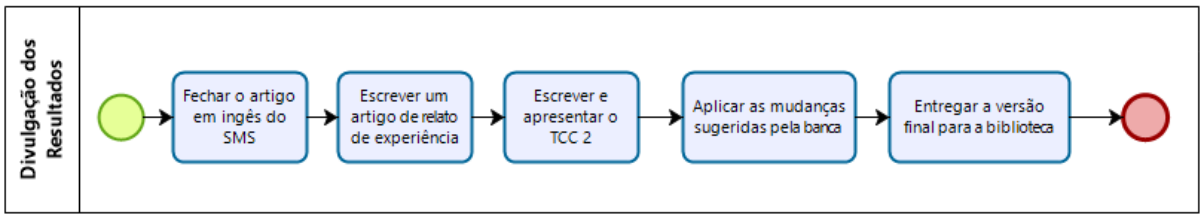


Figura 9 – Subprocesso para Divulgação dos Resultados no TCC 2

Fonte: O autor.

objetivo principal de explorar uma cadeia completa de ferramentas no suporte para a geração de solução de integrações em OSLC por meio de MDE. A Em adição, derivou-se um processo para a modelagem e o desenvolvimento dos adaptadores e um tutorial. Portanto, perdeu-se força no estudo se considerado um lado de avaliação, mas ganhou-se força por um lado mais metodológico, um vez que o resultado pode agora contribuir para a redução da curva de aprendizado no tema.

Por fim, o último subprocesso mostrado na Figura 9 trata da divulgação dos resultados do TCC 2.

2.3 Lições do Capítulo

A pandemia de COVID-19 apresentou um desafio para os planejamentos de avaliação discutidas no TCC 1. Inicialmente, se previa a transferência de conhecimento sobre integração de aplicação para uma equipe do DTIC. Tal previsão foi alinhada no TCC 1 como a **Atividade 4** tratando da execução do estudo de caso no DTIC, avaliando a solução de integração, e incluía as seguintes subatividades: **Atividade 4.1:** Revisar e refinar os adaptadores de *toolchain* para as ferramentas identificadas no *survey*, colocando em operação uma solução de integração para o DTIC em um servidor Web. **Atividade 4.2:** Executar o estudo de caso. Para tal, pretendia-se elaborar um questionário com times. **Atividade 4.3:** Analisar e relatar os resultados do estudo de caso.

A metodologia precisou ser totalmente adaptada para o novo cenário. Assim, nenhuma destas atividades foi executada. Pretendia-se executar o estudo de caso planejado e relatar sobre fatores de sucesso/falha, benefícios/malefícios e demais observações, na utilização de OSLC, na visão da equipe do DTIC, sobre a integração das ferramentas de produção de software.

No lugar, adotou-se um estudo que explora um cenário do DTIC, cuja experiência no desenvolvimento e o relato de aprendizado depende apenas da visão do pesquisador. Este novo protocolo é discutido como um relato de experiência, como detalhado em um segundo artigo que segue no Apêndice B, submetido para o Simpósio Brasileiro de Qualidade de Software (SBQS 2020).

O relato de experiência foi construído a partir dos benefícios e malefício extraídos

no SMS. Eles foram agrupados em atributos de qualidade que puderam ser avaliados no cenário anteriormente apresentado. Por fim, identificou-se o aparato ferramental utilizado no estudo, focadas em ferramentas para auxiliar as atividades da disciplina de Engenharia de Software, comparando-o com o aparato ferramental identificado nos estudos do SMS. Com isso, posiciona-se em relação aos demais estudos exploratórios, demonstrando que a área de pesquisa também utiliza deste tipo de avaliação para comprovar suas contribuições.

3 EMBASAMENTO TEÓRICO

Este capítulo aborda os temas relacionados a pesquisa realizada durante o desenvolvimento deste trabalho.

3.1 Integração de Ferramentas

A área de estudo sobre integração de ferramentas possuem contribuições desde a década de 1980. Wasserman (WASSERMAN, 1990) propõe em seu estudo cinco níveis de integração de ferramentas: (i) plataforma, (ii) controle, (iii) processo, (iv) dados e (v) apresentação. Relacionado a integração de dados, existem desafios para a troca de artefatos entre ferramentas que envolvem manter a consistência e a rastreabilidade dos dados.

Integração de ferramentas em ambientes de ES trata sobre como ferramentas com características heterogêneas podem ser compatíveis em diferentes aspectos. Esses aspectos incluem os formatos dos dados, convenções de interface do usuário, funcionalidades da aplicação e outros aspectos do desenvolvimento da ferramenta (THOMAS; NEJMEH, 1992). Ainda que o maior desafio seja prover um ambiente integrado de forma automatizada que compreenda todo o ciclo de vida do software (BROWN; PENEDO, 1992), existem diversas abordagens para integrar ferramentas de software. Cada uma dessas abordagens envolve uma série de características que devem ser levadas em consideração durante sua escolha (HOHPE, 2003).

Nesse sentido, para se estabelecer uma integração entre aplicações, é importante escolher topologias e estilos de integração. As topologias definem como as aplicações irão se comunicar enquanto os estilos estabelecem a forma de compartilhamento de informações e funcionalidades entre as aplicações. Por exemplo, por meio da topologia ponto a ponto é possível compartilhar dados e funcionalidades. Essa topologia é caracterizada por manter as ferramentas fortemente acopladas e pelo crescimento exponencial do número de conexões conforme mais ferramentas são integradas. Isto significa que quanto mais ferramentas são adicionadas ao processo de integração, maior torna-se o custo operacional para as organizações.

3.2 *Application Lifecycle Management*

O ALM é um paradigma para integrar e gerenciar atividades relacionadas à governança, desenvolvimento e manutenção de produtos de software (TüZüN et al., 2019). Assim, as atividades do desenvolvimento de software são interligadas a outras etapas do processo, fornecendo a estrutura necessária para que todo o esforço de desenvolvimento de software, incluindo detalhes específicos sobre tarefas, funções, responsabilidades e marcos essenciais que ajudam as partes interessadas (desenvolvedores, engenheiros de software, clientes) a acompanhar o progresso do processo seja integrado (AIELLO, 2016).

Além disso, o ALM adota a ideia da integração ponta a ponta do ciclo de vida da aplicação. Dessa forma, o ambiente é integrado desde as atividades do primeiro contato com o cliente até a implantação do software. Por exemplo, um pedido de solicitação de mudanças do cliente pode ser associado aos requisitos do produto, e assim, promover a colaboração entre diferentes áreas do projeto.

Existem três características principais pelo qual o ALM é motivado: rastreabilidade, visibilidade e automação de processos (SCHWABER et al., 2006). Rastreabilidade é a capacidade de ligar os artefatos produzidos durante o projeto, importante para evitar inconsistência entre as múltiplas versões dos artefatos. A respeito da visibilidade, essa característica fornece as organizações a possibilidade de acompanhar o progresso das tarefas desenvolvidas pelas equipes do projeto. Para tal, é recomendado a automação das atividades do processo para que se consuma menos tempo e torne o processo mais eficaz, comparado com a execução das atividades manualmente.

3.3 *Open Services for Lifecycle Collaboration*

O OSLC é uma especificação industrial para representar ativos de software ALM intercambiáveis. Conforme ilustra a Figura 10, esse padrão é caracterizado pela topologia de integração ponto a ponto, onde todas as ferramentas implementam a mesma interface de comunicação definida como *Asset Management Specification* (AMS) (AMS, 2014). O OSLC define uma forma comum de representação dos artefatos criados durante o projeto, bem como funções que permitem outras aplicações manipulá-los. Dessa forma, ferramentas de software podem acessar os artefatos produzidos por outras ferramentas, desde que ambas possuam suporte ao OSLC.

Essas especificações consistem em regras e padrões para o uso do protocolo *Hypertext Transfer Protocol* (HTTP) e da estrutura de arquivos *Resource Description Framework* (RDF). Além da especificação principal, existem os domínios OSLC, no qual estendem a especificação principal e definem regras para seus respectivos domínios (Gerenciamento de Requisitos, Gerenciamento de Qualidade, Gerenciamento de Configuração e Mudanças, etc.). A Figura 11 apresenta a representação do modelo da especificação principal e a seguir são apresentados cada um dos elementos que compõem o diagrama:

Service Provider Catalog: Mantém um catálogo de *Services Providers*, podendo haver um catálogo principal redirecionando para outros *Services Providers Catalog*.

Service Provider: Representação de um produto, equipe, projeto ou serviço online que contém um conjunto de serviços OSLC.

Service: Um conjunto de artefatos gerenciados pelas ferramentas associadas a um domínio.

Delegated UI Dialog: É uma técnica em que um provedor pode incorporar uma interface de usuário de criação ou seleção a outro usando uma combinação de um código HTML e JavaScript.

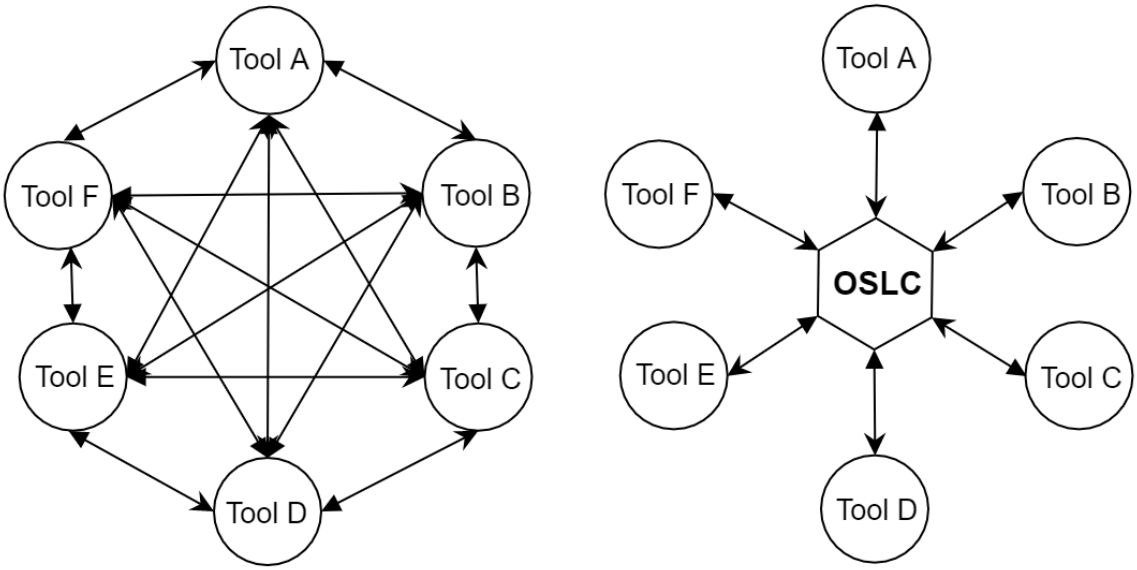


Figura 10 – Integração ponto a ponto com o padrão OSLC

fonte: O autor.

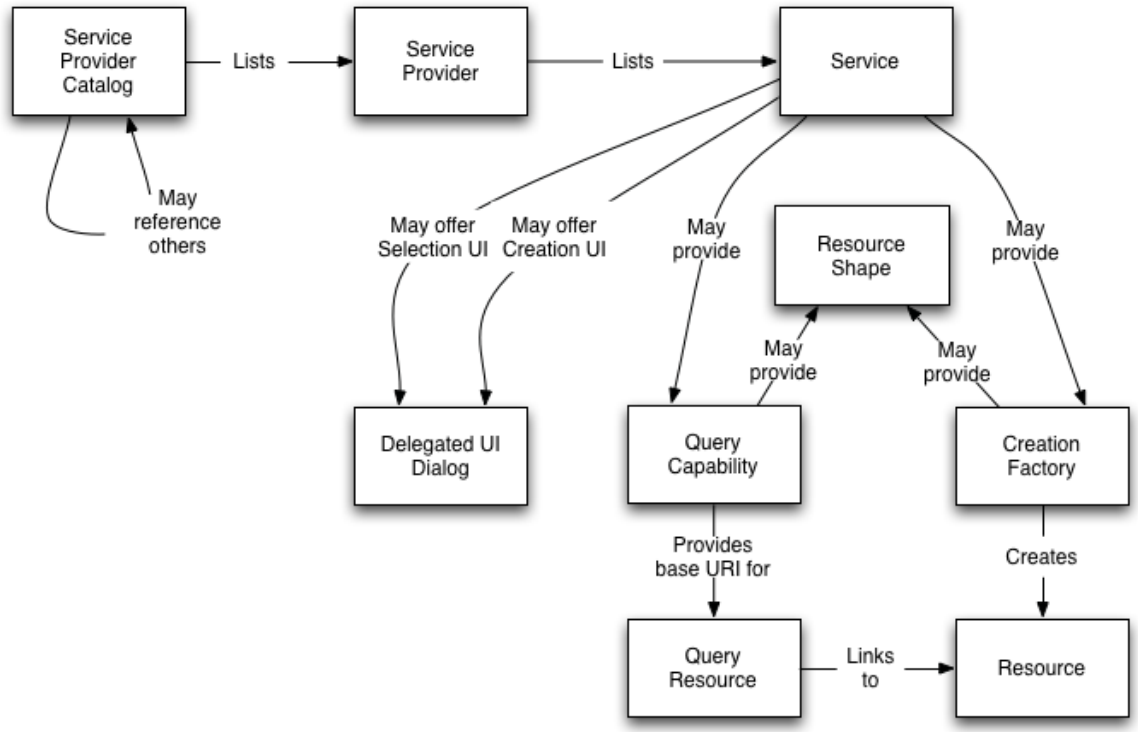


Figura 11 – OSLC Core Specification

Fonte: (OSLC, 2020b).

Regras Dados Ligados	Contexto OSLC
1. Use URIs (*Uniform Resource Identifier*) como nome pra coisas	Todo artefato pode ser representado por uma URI;
2. Use HTTP URIs para que as pessoas possam procurar esses nomes	Os URIs devem ser legíveis. Exemplo: http:example/requisito1 e não http:example/182837;
3. Quando alguém procura um URI, forneça informações úteis, utilizando os padrões RDF* ou SPARQL	Quando uma URI é requisitada via HTTP, o conteúdo do artefato é representado em formato RDF (geralmente);
4. Inclua *links* para outros URIs para que eles possam descobrir mais coisas.	Todos relacionamentos de um determinado artefato é representado por URIs (links) em seu conteúdo, tornando possível os dados ligados.

Tabela 1 – Dados Ligados no Contexto OSLC

Fonte: O autor.

Resource: Representação de um artefato de software (requisitos, solicitações de mudança, casos de teste, etc.) por meio do OSLC. Os *Resources* são representados e acessados por meio de uma URI.

Resource Shape: Define um conjunto de propriedades OSLC de um artefato como cada tipo de valor, valores permitidos, cardinalidade e opcionalidade.

Query Capability: Possibilita a consulta de OSLC Resources por meio da *Uniform Resource Identifier* (URI)).

Creation Factory Possibilita a criação de novos artefatos OSLC via funções HTTP *Post*.

Query Resource: Um parâmetro para consultas HTTP *Get*.

Com base nos princípios dos Dados Ligados, o OSLC segue as quatro regras definidas por Tim Beners-Lee para ligar dados na Web. A Tabela 1 mostra cada uma dessas regras equivalentes ao contexto OSLC. Os dados ligados são representados em arquivos RDF em uma estrutura chamada Tripla. Essa estrutura é composta por três itens: Sujeito, Predicado e Objeto. Os Sujeitos e Objetos podem ser um artefato ou uma pessoa que está realizando uma atividade, enquanto que o Predicado descreve um relacionamento entre Predicado e Objeto, como por exemplo artefato A pertence ao artefato B.

Em uma cadeia de ferramentas OSLC, uma aplicação pode ser classificada como Provedora ou Consumidora. Provedores destinam-se a armazenar e fornecer dados, permitindo aos consumidores acesso fácil para navegar, criar e recuperar dados (AICHERNIG et al., 2014).

Existem três abordagens para prover suporte OSLC em ferramentas de software: nativa, plugin e adaptador. A abordagem nativa é recomendada para fabricantes de ferramentas que desejam que suas ferramentas possuam suporte ao OSLC. As abordagens plugin e adaptador são semelhantes, entretanto a construção de plugins são recomendadas

apenas nos casos em que há o conhecimento das tecnologias que envolvem a construção da ferramenta, como linguagem de programação, arquitetura, etc. Em outros cenários, a abordagem recomendada é a construção de adaptadores OSLC.

3.4 Linguagem de Domínio Específico

DSL é uma linguagem de programação de computador com expressividade limitada, focada em um domínio específico. Diferente de uma linguagem de programação, DSLs são utilizadas em um domínio bem definido, com foco em resolver um problema central. O termo DSL é um conceito utilizado na área da computação há muitos anos e, atualmente, ainda existem discussões sobre a definição de DSL. Entretanto, existem algumas características que abrangem seu conceito, entre elas a de que DSLs devem ser legíveis por seres humanos.

Existem duas formas de definir um domínio específico: A primeira forma é por equações matemáticas, como por exemplo a criação de uma DSL para solucionar equações diferenciais e a segunda forma é pelo mapeamento de atividades humanas. Alguns exemplos são: construção de veículos e integração de ferramentas de software.

De acordo com Fowler (FOWLER; PARSONS, 2011), as DSLs são divididas em três categorias principais: (i) internas, (ii) externas e (iii) *language workbenches*. Apesar de possuírem o mesmo objetivo, cada uma dessas categorias possuem suas próprias características: DSLs internas possuem sua própria sintaxe e geralmente são construídas com base em *Extensible Markup Language* (XML). DSLs externas são baseadas em alguma linguagem existente e são vistas como uma extensão da linguagem utilizada. A respeito da categoria *language workbench*, esta envolve um Ambiente de Desenvolvimento Integrado (IDE) para a construção da DSL e sua sintaxe é construída com base nas características da própria IDE utilizada e na linguagem de programação escolhida para seu desenvolvimento. Além disso, a construção de uma DSL é diferente em cada uma dos três tipos, podendo um especialista em DSLs internas pode não saber desenvolver uma DSL externa. Fowler ainda argumenta que a utilização de DSLs emprega o aumento na produtividade e melhora a comunicação entre os especialistas do domínio, pois fica mais visível o que está acontecendo no sistema. Apesar dos benefícios, o uso de DSLs aumentam os gastos do projeto, principalmente pelo custo de sua manutenção.

3.5 O Estado da Arte em OSLC

OSLC, sendo um padrão aberto para interoperabilidade de ferramenta, permite a federação de dados em todo o ciclo de vida do software. A comunidade OSLC está ativa desde 2008 e ainda há uma questão em aberto: “Qual é o estado da arte e a prática OSLC para integrar ferramentas em atividades de ALM para ambientes de Engenharia de Software?”

Com o objetivo de caracterizar o estado da arte e a prática da área, o objetivo principal da fundamentação teórica foi mapear todas as abordagens que adotam OSLC nos ciclos de vida da ES. Assim, este trabalho de conclusão de curso apresenta na sequência um Estudo de Mapeamento Sistemático (SMS), analisando 59 estudos primários e abordando questões de integração, especialmente para cadeias de ferramentas SE nos fundamentos discutidos anteriormente.

Este estudo é detalhado no Apêndice A e evidencia descobertas mostram que o OSLC tem sido aplicado principalmente ao setor crítico de segurança. Também que ele é frequentemente usado nos domínios de Requisitos, Análise & Projeto, Gerenciamento de Configuração e Mudanças, e Testes.

Assim, enumerou-se as principais vantagens do OSLC estão relacionadas ao termo *Linked Data*, envolvendo não apenas adaptadores de ferramentas para integrações ponto a ponto, mas também propondo abordagens para substituir ferramentas no kit de ferramentas. Outra vantagem é a possibilidade de se modificar as especificações de domínio OSLC e soluções para atividades automatizadas para integração de ferramentas.

3.6 Estabelecendo uma Ponte Conceitual

As ferramentas discutidas no contexto deste trabalho de conclusão de curso são utilizadas para apoiar profissionais na execução de atividades ao longo do ciclo de vida do software. Essas ferramentas possuem diferentes funcionalidades, configurações, fabricantes e são desenvolvidas sem o suporte nativo para integração com outras utilizadas nos contextos de ES. Além disso, manipulam artefatos (requisitos de software, modelos, código-fonte, casos de teste, etc.) produzidos em diferentes fases do desenvolvimento. Assim, Wicks (WICKS; DEWAR, 2007) argumenta que este cenário contribui para que ambientes integrados de ponta a ponta de forma totalmente automatizada sejam raros na indústria.

Organizações de software obtêm novas ferramentas ou desenvolvem suas soluções de software ao longo dos anos, resultando em um ecossistema de software heterogêneo (MESSERSCHMITT, 2005). Isso demanda que as organizações adotem abordagens para melhoria dos processos de software e suporte ferramental que permitam a automação do processo de desenvolvimento por meio da integração de ferramentas.

Entre as alternativas está o ALM, o qual adota a ideia de integração ponta a ponta, evitando silos de informações devido a características como rastreabilidade, visibilidade e automação de processos (SCHWABER et al., 2006). Para reduzir os problemas relacionados à integração de artefatos de software ao longo de várias fases do ALM, padrões de representação para modelos, metamodelos e transformações têm sido propostas na literatura (KLEPPE; WARMER; BAST, 2003). Além disso, para mitigar a complexidade desses ambientes, várias ferramentas que auxiliam nas tarefas de Engenharia de Software foram propostas (EBERT, 2013). Essas ferramentas ajudam os desenvolvedores a apro-

veitar experiências anteriores ao especificar um novo software, enquanto ferramentas com base em modelos ajudam a capturar, organizar e armazenar a maioria das informações trocadas como ativos de software reutilizáveis.

Para ajudar na integração de ferramentas usadas na produção de software, muitas abordagens foram propostas interoperando dados entre serviços (BIEHL et al., 2014). As abordagens foco destes trabalhos caracterizam-se por minimizar os problemas relacionados ao processo de integração. Elas são adotadas nos contextos de ES, tipicamente implementadas por meio de uma arquitetura comum para serviços (ZHANG; MøLLER-PEDERSEN, 2014), o que demanda um protocolo de comunicação comum entre diferentes ferramentas.

Em adição, diversas abordagens têm sido propostas para a *Continuous Software Engineering (CSE)*, em especial aquelas que integram ferramentas com o padrão OSLC. Estabelecer cadeias de ferramentas OSLC exige dos engenheiros de software o conhecimento de como criar interfaces de ferramentas para a troca de dados interoperada por meio de adaptadores. Essa tarefa não é fácil e requer etapas bem definidas que ajudam os engenheiros de software na execução de atividades que garantam a realização da engenharia de software contínua.

3.7 Lições do Capítulo

Este capítulo apresentou uma fundamentação teórica para o acompanhamento dos resultados de uma pesquisa exploratória. Uma vez que se explora uma ferramenta construída por terceiros, e fundamentada nos conceitos de *Model Driven-Development (MDD)*, alinhou-se os conceitos para a integração de ferramentas e para o padrão de serviços denominado OSLC. A abordagem MDD permite a geração de código-fonte de soluções de integração, o que pode ajudar na assimilação da teoria da integração e prática.

Além disso, tal ferramenta também inclui um plugin de modelagem de solução de integração. Uma vez que o uso de *Domain-Specific Language (DSL)* aumenta a produtividade e melhora a comunicação entre os especialistas do domínio, explora-se estas características no suporte ferramental investigado.

Por fim, o capítulo estabeleceu uma ponte conceitual entre os vários termos que estão associados com a pesquisa conduzida neste trabalho de conclusão de curso. Para maiores informações abordando elementos conceituais, recomenda-se a leitura dos dois artigos derivados deste trabalho, que constam no Apêndice A e Apêndice B. A seguir, apresenta-se um seguimento da fundamentação teórica, com um estudo de mapeamento sistemático conduzido no tema.

4 TRABALHOS RELACIONADOS

Este Capítulo apresenta a busca por trabalhos relacionados na literatura. Para isso, realizou-se um SMS, no qual 59 artigos foram avaliados com o objetivo de identificar o estado da arte e prática do uso do OSLC em ambientes de ES. Este trabalho segue um protocolo bem definido para o planejamento e execução de mapeamentos sistemáticos na área de ES (PETERSEN; VAKKALANKA; KUZNIARZ, 2015).

4.1 Protocolo do SMS

4.1.1 Questões de Pesquisa

As questões de pesquisa foram estruturadas com o propósito de identificar quais são as características que motivam o uso do padrão OSLC ao longo do ciclo de vida do software. As questões de pesquisa são apresentadas a seguir:

Q1. *Quais as fases do ciclo de vida do software o padrão OSLC é utilizado?* Os estudos de integração de aplicações direcionados ao contexto de ES geralmente estão associados a determinados ciclos de vida do software. Assim, esta questão de pesquisa procura caracterizar os estudos que contribuem para determinadas disciplinas de ES, fornecendo um mapa de cobertura que permite inferir se os processos de desenvolvimento de software são totalmente automatizados.

Q2. *Quais são os contextos das organizações que usam o padrão OSLC?* Para fornecer um mapa do estado da prática sobre o tema, também é do interesse deste estudo entender que tipo de organização está adotando o padrão OSLC. Assim, espera-se estabelecer uma relação entre teoria e prática na área de pesquisa.

Q3. *Quais são os tipos mais frequentes de pesquisa sobre o padrão OSLC?* Para entender a maturidade dos estudos do ponto de vista empírico, busca-se também classificá-los de acordo com o tipo de avaliação adotada.

Q4. *Quais são as facetas da contribuição do estudo?* Por fim, o objetivo é caracterizar as contribuições da área de acordo com sua faceta: seja com ferramentas, modelagem, processos e metodologias.

4.1.2 Estratégia de Busca

A busca pelos artigos foi realizada em 5 bibliotecas digitais amplamente utilizadas pela comunidade científica: ACM Digital Library[1], IEEEXplore[2], Springer Link[3], Science Direct[4] e Scopus[5]. As *strings* de busca foram elaboradas com o objetivo de obter uma ampla cobertura de estudos sobre o padrão OSLC, como mostra a Tabela 2.

[1] https://dl.acm.org/
[2] https://ieeexplore.ieee.org/
[3] https://link.springer.com/
[4] https://www.sciencedirect.com/
[5] https://www.scopus.com/

Base de Dados	*String* de Busca
ACM DL	("Open Services for Lifecycle Collaboration" OSLC) AND (Integration Interoperability "Linked Data" Lifecycle Traceability)
IEEE Xplore	("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability)
Springer Link	("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability)
Science Direct	("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability)
Scopus	TITLE-ABS-KEY (("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability))

Tabela 2 – *Strings* de Busca

Fonte: O autor.

4.1.3 Critérios de Inclusão e Exclusão

Os Critérios de Inclusão (CI) e Critérios de Exclusão (CE) possuem o objetivo de filtrar os trabalhos relevantes para a pesquisa. Para o estudo ser selecionado é necessário que ele satisfaça todos os CI. No entanto, se o artigo estiver associado a pelo menos um CE, ele é excluído do mapeamento. Os critérios definidos são os seguintes:

CI1. O estudo apresenta no título, resumo, palavras-chave ou introdução os termos OSLC ou *Open Services for Lifecycle Collaboration*.

CI2. O estudo apresenta um exemplo de uso, método, ferramenta ou processo que utiliza o padrão OSLC como solução.

CE1. O estudo não é um artigo escrito em inglês.

CE2. O estudo não é primário.

CE3. O estudo possui menos de quatro páginas.

CE4. O estudo não é da área da Ciência da Computação.

CE5. O estudo foi publicado anterior ao ano de 2008.

CE6. O estudo não está disponível para download.

CE7. O estudo é duplicado entre as bases de dados.

4.1.4 Critérios de Avaliação de Qualidade

Os Critérios de Avaliação de Qualidade (CQA) têm o objetivo de classificar os artigos de acordo com a sua relevância para o trabalho. Os artigos recebem uma nota para cada CQA e podem ser classificados como Qualidade Excelente, Qualidade Muito Boa, Qualidade Boa, Qualidade Parcial e Qualidade Baixa. Os Critérios de Avaliação de

Qualidade são os seguintes:

CQA1. Aplicabilidade para ciclos de vida de Engenharia de Software: Aplica-se a estimar a soma das contribuições apresentadas pelo artigo referente à questão de pesquisa **Q1**.

- **Qualidade Excelente**: Explora a integração de cinco ou mais ferramentas de pelo menos cinco fases diferentes do ciclo de vida de Engenharia de Software. Valor 3.
- **Qualidade Boa**: Explora a integração de três ferramentas de três fases diferentes do ciclo de vida de Engenharia de Software. Valor 2.
- **Qualidade Parcial**: Explora a integração de apenas três ou duas ferramentas adotadas em dois ciclos de vida de Engenharia de Software. Valor 1.
- **Qualidade Baixa**: Não explora os ciclos de vida de Engenharia de Software. Valor 0.

CQA2. Relatório de Viabilidade: Aplica-se à questão de pesquisa **Q2** e visa identificar a relevância do relatório, considerando as necessidades reais de organizações.

- **Qualidade Excelente**: O estudo apresenta um relatório completo de viabilidade em um cenário organizacional real. Valor 2.
- **Boa Qualidade**: O estudo apresenta um relatório de viabilidade em um cenário organizacional também real. Valor 1.
- **Baixa Qualidade**: O estudo não apresenta relatório de viabilidade. Valor 0.

CQA3. Metodologia de avaliação que extraiu suas conclusões: Aplica-se ao tipo de avaliação de referente à questão de pesquisa Q3.

- **Qualidade Excelente**: O estudo é um *evaluation research*, ou seja, apresenta resultados de um experimento controlado, ou de um estudo de caso ou de uma pesquisa-ação. Valor 3.
- **Qualidade Muito Boa**: O estudo é do tipo *validation research*, ou seja, apresenta resultados de um estudo analítico, de uma pesquisa, de uma simulação ou de um grupo focal. Valor 2.
- **Qualidade Boa**: O estudo é do tipo *solution proposal*, ou seja, suas conclusões são derivadas de uma prova de conceito, também conhecida como demonstração conceitual de uma ferramenta proposta, método ou processo. Valor 1.
- **Qualidade Parcial**: O estudo é do tipo *experience report*, com o relato dos próprios autores sobre o contexto OSLC. Valor 0,5.

Item	**Questão de Pesquisa**
Id do Artigo	-
Título do Artigo	-
Fases do Ciclo de Vida do Software	Q1
Ferramentas	Q1
Processo de Desenvolvimento de Software	Q1
Vantagens	Q1
Desvantagens	Q1
Contexto da Organização	Q2
Tipo de Pesquisa	Q3
Faceta da Contribuição	Q4

Tabela 3 – Formulário de Extração de Dados

fonte: O autor.

- **Qualidade Baixa**: O estudo é um trabalho *philosophical paper* do tipo *opinion paper*. Valor 0.

CQA4. Faceta de contribuição no suporte de ferramentas MDE: Aplica-se a estimar as contribuições que fornecem suporte de ferramentas para representação de integração independente das plataformas OSLC, ou seja, o peso é cumulativo para a pergunta de pesquisa **Q4**.

- **Qualidade Excelente**: Apresenta DSL e modelo próprios de transformações para gerar especificações OSLC. Valor 2.
- **Qualidade Muito Boa**: Apresenta uma DSL de terceiros e transformações próprias de modelo para gerar especificações DSL. Valor 1.5.
- **Qualidade Boa**: Apresenta uma DSL própria. Valor 1.
- **Qualidade Parcial**: Apresenta uma DSL de terceiros. Valor 0.5.
- **Qualidade Baixa**: Não fornece uma DSL para o *design* de integração. Valor 0.

4.1.5 Estratégia de Extração dos Dados

Para responder às questões de pesquisa apresentadas e analisar adequadamente seus resultados, as informações dos trabalhos foram registradas no formulário de extração de dados apresentado na Tabela 3.

4.2 Execução

Primeiramente, as *strings* de busca foram aplicadas nas bases de dados, retornando 278 artigos, como mostra a Figura 12. Em seguida, os critérios de inclusão e exclusão

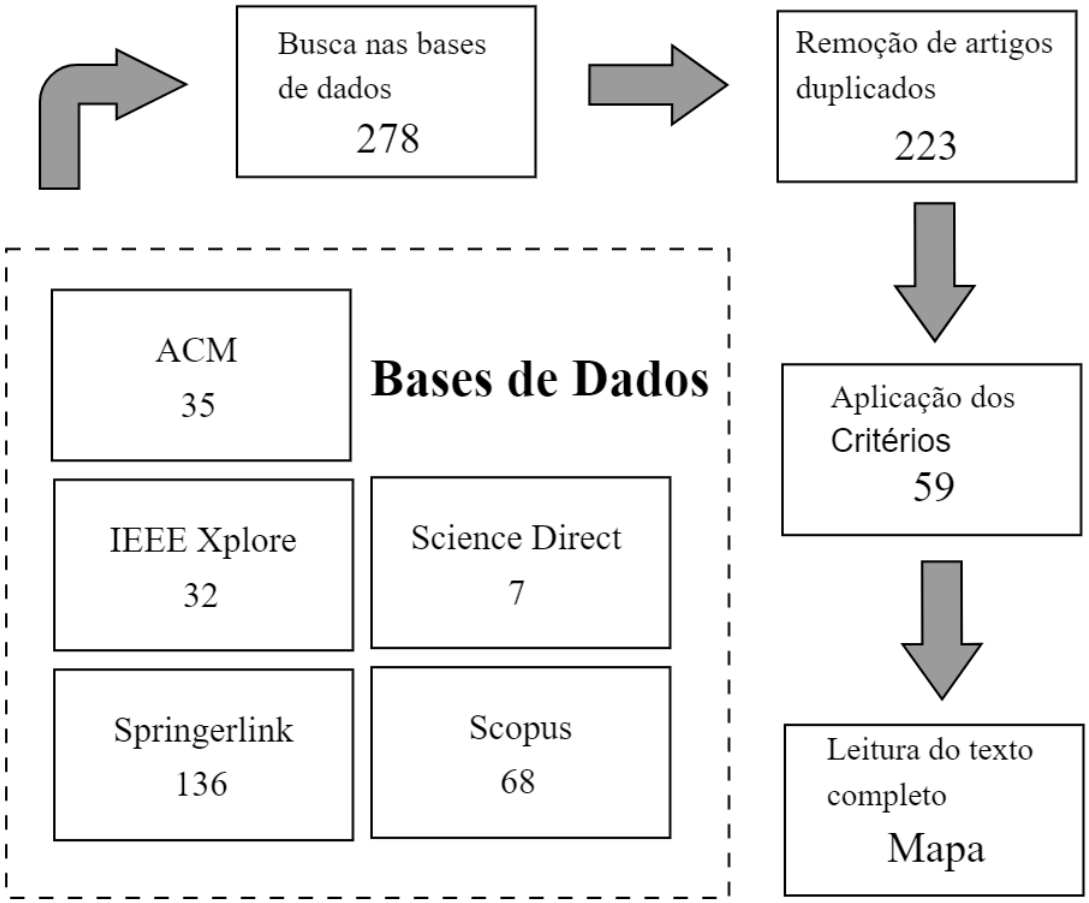


Figura 12 – Processo de Triagem dos Estudos

Fonte: O autor.

foram empregados com base no título, resumo, palavras-chave e introdução. Como resultado, 59 estudos foram considerados acordo com o tópico da pesquisa, sendo eles: 7 artigos da IEEE Xplore, 1 da Springerlink, 3 da Science Direct, 48 da Scopus e nenhum da ACM. A atividade de seleção dos artigos foi realizada na ferramenta online Parsif.al[6], a qual possibilita a organização dos artigos e evita a possibilidade que trabalhos sejam esquecidos durante o processo.

4.3 Análise dos Resultados

A Tabela 4 mostra a lista de todos os estudos selecionados após o processo de triagem e sua relevância para a pesquisa. O primeiro trabalho encontrado foi publicado em 2010 e relata a pesquisa e o potencial de uma nova especificação para integração de ferramentas. Desde então, pesquisadores exploraram o OSLC e propuseram novas abordagens para aprimorar a integração de ferramentas em ambientes ALM.

[6] https://parsif.al/

ID	Referência do Estudo	CQ1	CQ2	CQ3	CQ4	Nota Final
s1	(BAUMGART; ELLEN, 2014)	2	2	1	0.5	5.5
s2	(MARKO et al., 2015)	2	2	1	0	5
s3	(VANZANDT, 2015)	0	2	0	0	2
s4	(KLESPITZ; BíRó; KOVáCS, 2016b)	0	1	0	0	1
s5	(LEDNICKI et al., 2016)	1	2	1	0.5	4.5
s6	(ZHANG; MøLLER-PEDERSEN, 2014)	2	2	1	2	6
s7	(MARTINO et al., 2016)	0	1	1	0	2
s8	(EL-KHOURY, 2016)	0	1	0	2	3
s9	(SHARIATZADEH et al., 2016a)	2	1	1	0	4
s10	(GüRDüR et al., 2018)	0	2	2	0.5	4.5
s11	(NARDONE et al., 2020)	1	2	1	0.5	4.5
s12	(EL-KHOURY; BEREZOVSKYI; NYBERG, 2019)	2	2	1.5	0.5	6
s13	(TROUBITSYNA, 2019)	0	0	0	0	0
s14	(KERN et al., 2019)	1	2	1	0.5	4.5
s15	(CHEN et al., 2019)	0	1	1	0	2
s16	(BEREZOVSKYI; EL-KHOURY; FERSMAN, 2019)	0	1	1	0.5	2.5
s17	(PIKUS et al., 2019)	1	2	1	0	4
s18	(LU et al., 2018)	1	2	1	2	6
s19	(ARNOULD, 2018)	0	1	1	1	3
s20	(MUSTAFA; LABICHE, 2018)	1	2	3	1	6
s21	(ALVAREZ-RODRíGUEZ et al., 2018)	1	2	2	0	5
s22	(BUFFONI; POP; MENGIST, 2017)	1	1	1	0	3
s23	(GALLINA; NYBERG, 2017)	2	1	0	0	3
s24	(GALLINA; PADIRA; NYBERG, 2016)	0	1	2	0	3
s25	(VAGLIANO et al., 2017)	1	2	1	0.5	4.5
s26	(KLESPITZ; BíRó; KOVáCS, 2016a)	0	0	0	0	0
s27	(MUSTAFA; LABICHE, 2017)	0	0	1	0	1
s28	(BIRÓ et al., 2017)	2	1	1	0.5	4.5
s29	(ZADEH et al., 2017)	1	2	3	0.5	6.5
s30	(RENé; MARTIN, 2017)	0	1	1	0.5	2.5

Tabela 4 – Lista dos Estudos Selecionados e Aplicação dos Critérios de Qualidade Parte I

fonte: O autor.

ID	**Referência do Estudo**	CQ1	CQ2	CQ3	CQ4	Nota Final
s31	(ILIASOV et al., 2016)	0	2	1	0.5	3.5
s32	(LEITNER; HERBST; MATHIJSSEN, 2016)	0	2	0.5	0.5	3
s33	(BIRÓ et al., 2016)	0	1	0	0	1
s34	(SHARIATZADEH et al., 2016b)	0	1	1	0	2
s35	(EL-KHOURY; EKELIN; EKHOLM, 2016)	1	2	1	1.5	5.5
s36	(GIANDOMENICO et al., 2016)	3	2	3	0.5	8.5
s37	(ADJEPON-YAMOAH; ROMANOVSKY; ILIASOV, 2015)	0	2	1	0.5	3.5
s38	(KAISER; HERBST, 2015)	0	1	0	0	1
s39	(REGAN et al., 2015)	0	1	2	0	3
s40	(SHAHROKNI; SöDERBERG, 2015)	0	2	0.5	0	2.5
s41	(AICHERNIG et al., 2014)	1	2	1	0	4
s42	(SIEBENMARCK, 2014)	0	0	0	0	0
s43	(AOYAMA et al., 2014)	1	2	1	0.5	4.5
s44	(BIRO, 2014)	0	0	0.5	0	0.5
s45	(SAADATMAND; BUCAIONI, 2014)	0	1	0	0.5	1.5
s46	(KACIMI et al., 2014)	2	2	1	0	5
s47	(WOLVERS; SECELEANU, 2013)	2	2	1	0.5	5.5
s48	(ELAASAR; NEAL, 2013)	1	2	2	1	6
s49	(BIEHL et al., 2013)	0	2	1	1.5	4.5
s50	(ELAASAR; CONALLEN, 2013)	0	0	1	1	2
s51	(AOYAMA et al., 2013)	1	2	0	0.5	3.5
s52	(RYMAN; HORS; SPEICHER, 2013)	0	0	0	0	0
s53	(ZHANG; MøLLER-PEDERSEN, 2013)	2	2	0	0.5	4.5
s54	(ZHANG; MøLLER-PEDERSEN; BIEHL, 2012)	2	2	1	2	7
s55	(BIEHL; EL-KHOURY; TöRNGREN, 2012)	2	1	1	2	6
s56	(BIEHL; GU; LOIRET, 2012)	0	0	1	0.5	1.5
s57	(ZHANG et al., 2012)	2	2	1	2	7
s58	(BERGER et al., 2010)	0	0	0.5	0	0.5
s59	(REGAN et al., 2014)	0	1	1	0	2

Tabela 5 – Lista dos Estudos Selecionados e Aplicação dos Critérios de Qualidade Parte II

fonte: O autor.

Q1. Quais as fases do ciclo de vida do software o padrão OSLC é utilizado?

Para responder a **Q1**, foram identificados três pontos dos estudos: (i) as fases do ciclo de vida do software nas quais o OSLC é frequentemente usado, (ii) as ferramentas de software associadas às soluções propostas, e o (iii) processo de desenvolvimento de software adotado pelos autores.

As disciplinas do *Rational Unified Process* (RUP) foram usadas como categorias para mapear o uso do OSLC. Essa decisão se deve ao fato de muitos engenheiros de software terem um conhecimento prévio sobre essas disciplinas. A seguir, são descritas cada uma das terminologias adotadas para essas disciplinas de Engenharia de Software.

- Modelagem de Negócios: tem como objetivo descrever como o produto será desenvolvido, levando em consideração os processos, funções e clientes da organização.
- Requisitos: o objetivo desta disciplina é definir o escopo do sistema a partir das solicitações das partes interessadas.
- Análise & Projeto: mostra como o sistema será desenvolvido a partir de modelos de tarefas, código-fonte ou componentes de software.
- Implementação: essa disciplina é composta por atividades para desenvolver código-fonte, como classes e objetos.
- Teste: é dedicado a garantir a qualidade do produto, incluindo a verificação e validação de ativos de software, como requisitos, diagramas etc.
- Implantação: essa disciplina se concentra nas atividades de entrega do software para seus usuários finais.
- Gerenciamento de Configuração e Mudanças: inclui atividades como o controle de versões e solicitações de mudança.
- Gerenciamento do Projeto: é dedicado a tarefas que envolvem o gerenciamento do projeto e o gerenciamento de riscos.
- Ambiente: concentra-se em atividades para fornecer um ambiente de desenvolvimento de software configurado e pronto para o trabalho.

De acordo com os resultados do SMS mostrados na Tabela 6, a maioria das soluções propostas são para os domínios de gerenciamento de requisitos, análise & projeto, testes e gerenciamento de configuração e mudanças. Todas as ferramentas na cadeia de ferramentas foram consideradas como parte da solução de integração OSLC. Portanto, os estudos podem ser categorizados em mais de uma disciplina. A Tabela 7 mostra a distribuição dos artigos em cada uma das disciplinas do RUP.

ID	Modelagem de Negócios	Requisitos	Análise & Projeto	Implementação	Teste	Implantação	Gerenciamento de Configuração e Mudanças	Gerenciamento de Projeto	Ambiente
s1		X	X		X				
s2		X			X				
s3							X		
s4								X	X
s5							X		
s6		X	X		X				
s7	X	X	X	X					
s8			X						
s9			X		X				
s10									
s11		X	X		X				
s12									
s13									
s14				X	X				
s15									
s16									
s17							X		
s18			X		X			X	
s19									
s20		X	X		X				
s21		X	X		X				
s22		X			X		X		
s23		X	X	X	X				
s24					X				
s25			X		X				
s26								X	X
s27			X						
s28		X	X	X	X				
s29			X						
s30		X	X						
s31		X							
s32									
s33		X	X	X	X				
s34									
s35			X						
s36		X	X		X		X	X	
s37							X		
s38		X	X		X				
s39							X		
s40									
s41		X			X				
s42									
s43							X	X	
s44									
s45					X				
s46		X	X		X				
s47		X	X	X	X				
s48			X						
s49		X			X				
s50			X				X		
s51							X	X	
s52							X		
s53		X	X		X		X		
s54		X	X		X				
s55			X		X				
s56			X						
s57			X						
s58							X		
s59							X		

Tabela 6 – Estudos Distribuídos pelas Disciplinas do RUP

Fonte: O autor.

As abordagens OSLC se empenham em resolver problemas comuns entre os trabalhos. No domínio dos requisitos, o OSLC é usado para estabelecer a rastreabilidade entre os requisitos e o restante do projeto. Além disso, as ferramentas de requisitos de software são, frequentemente, usadas nos exemplos de conjuntos de ferramentas como um ponto de partida para o fluxo de trabalho.

Por meio dos conceitos dos dados ligados, trabalhos como (VANZANDT, 2015), (BUFFONI; POP; MENGIST, 2017) propõe abordagens para gerenciar o histórico de mudanças, incluindo as partes interessadas que o modificaram e tarefas nas quais os artefatos estão associados. Nesse sentido, abordagens de domínio Gerenciamento de Configuração e Mudanças como (LEDNICKI et al., 2016), (PIKUS et al., 2019), (ADJEPON-YAMOAH; ROMANOVSKY; ILIASOV, 2015), (REGAN et al., 2015), (AOYAMA et al., 2014), (AOYAMA et al., 2013) são propostas para controle de versão e automação de tarefas.

Os resultados do SMS reforçam a necessidade de integrar diferentes domínios do

Disciplina do RUP	Quantidade
Modelagem de Negócios	1
Requisitos	21
Análise & Projeto	29
Implementação	6
Teste	25
Implantação	0
Gerenciamento de Configuração e Mudanças	14
Gerenciamento de Projeto	6
Ambiente	2
N/A	11

Tabela 7 – Quantidade de Estudos em Cada Disciplina do RUP

fonte: O autor.

desenvolvimento de software desde suas fases iniciais. Portanto, ferramentas para Engenharia Orientada a Modelos são citadas na fase de Análise & Projeto como soluções para minimizar esses esforços de integração. Por exemplo, o Eclipse Lyo (EL-KHOURY, 2016) propõe um gerador de código-fonte para interfaces de ferramenta em conformidade com OSLC no ambiente Eclipse, no qual foi citado pelos trabalhos no desenvolvimento de adaptadores OSLC. Nesse sentido, alguns trabalhos (MARTINO et al., 2016), (LU et al., 2018), (ARNOULD, 2018), (MUSTAFA; LABICHE, 2017), (ZHANG; MøLLER-PEDERSEN, 2013), (BIEHL; EL-KHOURY; TöRNGREN, 2012), (BIEHL; GU; LOIRET, 2012) propõe abordagens para apoiar a integração ou transformação de modelos heterogêneos, como *Business Process Model and Notation* (BPMN), *Unified Modeling Language* (UML), *Systems Modeling Language* (SysML), *Service oriented architecture Modeling Language* (SoaML) e *Eclipse Modeling Framework* (EMF).

Além disso, há um número considerável de soluções aplicadas à disciplina de teste devido aos contextos em que o OSLC é normalmente aplicado. Por tratarem-se de sistemas críticos, os conjuntos de ferramentas são compostos por ferramentas de Verificação e Validação, contribuindo para esses resultados.

A maioria dos estudos não estão associados a um processo específico de desenvolvimento de software. No entanto, em oito trabalhos (NARDONE et al., 2020), (KERN et al., 2019), (GALLINA; NYBERG, 2017), (GALLINA; PADIRA; NYBERG, 2016), (BIRÓ et al., 2017), (BIRÓ et al., 2016), (GIANDOMENICO et al., 2016), (KAISER; HERBST, 2015), o processo *V-model* foi mencionado pelos autores como o processo de desenvolvimento presente em o ambiente.

Por fim, os trabalhos selecionados relatam vantagens e desvantagens comuns a outros estudos. Por exemplo, as vantagens do OSLC são obtidas a partir de princípios de dados ligados. Por meio das especificações dos domínios OSLC em que os artefatos são representados, a rastreabilidade e a consistências são atingidos. Além disso, a colaboração entre diferentes equipes de desenvolvimento é aprimorada devido a troca de dados antes

não realizada entre aplicações.

Apesar das vantagens do uso do OSLC, os autores citam duas desvantagens principais: (i) a dificuldade de implantação do OSLC, seja pela falta de documentação ou pelo alto conhecimento técnico exigido; e (ii) o OSLC requer um esforço de desenvolvimento maior do que o exigido por uma integração de serviço da web simples, tornando uma tarefa custosa para as organizações.

Q2. Quais são os contextos das organizações que usam o padrão OSLC?

A maioria dos estudos encontrados são contextualizados em ambientes de sistemas críticos, como sistemas médicos, sistemas embarcados, indústria automotiva e indústria aeronáutica. Esses sistemas são caracterizados por conterem atividades de verificação e validação, como por exemplo simulações e diferentes níveis de testes. Em outros 13 trabalhos, o contexto do ambiente não é citado pelo autor. O contexto das categorias de organizações é explicado da seguinte forma:

- Medicina: Sistemas para ambientes médicos.
- Sistemas Embarcados: Sistemas de computação responsáveis por executar uma única tarefa específica.
- Academia: Desenvolvimento de sistemas em colaboração com universidades.
- Automotiva: Desenvolvimento de sistemas para o contexto automotivo.
- Sistemas Críticos: Sistemas que contém a execução de atividades de verificação e validação por não ser permitido falhas em sua execução por envolver vidas humanas ou grandes perdas financeiras.
- Sistema Ciber-físicos/Internet das Coisas: Ambientes que integram software e hardware.
- Aeronáutica: Desenvolvimento de sistemas para aeronaves.
- Realidade Aumentada: Sistemas de realidade virtual.
- Marinha: Desenvolvimento de sistemas para embarcações marítimas.

As empresas que forneceram a validação das especificações OSLC no setor são consideradas grandes organizações, envolvendo um número significativo de funcionários, como a montadora Scania. Devido aos requisitos impostos à operação desses sistemas, seja por normas governamentais ou ISOs, o OSLC é proposto como uma alternativa para estabelecer integrações na camada de dados das aplicações que compõem esses ecossistemas, principalmente nas atividades de teste.

Embora o OSLC atenda às necessidades desses ambientes organizacionais, os autores mostram que a integração de ambientes não é uma tarefa trivial. Além disso, a

integração desses ambientes requer profissionais especializados, o que pode aumentar o custo de sua adoção. Uma das opções que as empresas buscam é adotar abordagens como MDD e MDE para geração de interfaces integradoras. Este cenário pode ser uma alternativa para que empresas de médio e pequeno porte adotem o OSLC em seus ambientes.

Q3. Quais são os tipos mais frequentes de pesquisa sobre o padrão OSLC?

A maturidade de uma pesquisa pode ser determinada pela forma como o estudo afeta a prática na área. Por esse motivo, adotamos as categorias propostas pelo estudo de Wieringa *et al.* (WIERINGA et al., 2005), mapeando os tipos de pesquisa empregados por cada pesquisa. Assim, são considerados os seguintes tipos de estudos:

- *Evaluation research*: é qualquer trabalho que apresente um estudo empírico, como estudo de caso, estudo de campo, pesquisa etc.
- *Validation research*: é composta por estudos que apresentam pesquisas quantitativas, como experimentos, simulação, prototipagem, análise matemática, etc.
- *Solution proposal*: esses documentos apresentam uma prova de conceito, como o desenvolvimento de um pequeno exemplo ou um argumento sólido.
- *Philosophical paper*: qualquer artigo que proponha novas abordagens conceituais ou discussões de uma nova maneira de analisar as abordagens existentes.
- *Opinion paper*: estes artigos discutem as opiniões do autor sobre vantagens ou desvantagens dos tópicos de pesquisa, como o setor deve usar abordagens etc.
- *Experience report*: é composto por lições aprendidas na experiência do setor, como desafios em projetos, desenvolvimento de produtos, avaliação de abordagens etc.

A Tabela 8 mostra a maioria dos artigos classificados como *solution proposal* (36), reforçando a ideia de que OSLC é implementado como prova de conceito e protótipos em ambientes reais. Alguns trabalhos sugerem a realização de estudos de caso, embora sejam apresentados dados insuficientes, levando a um pequeno número de estudos empíricos, incluindo pesquisas do tipo *validation research* (3) e *evaluation research* (5). Entre os estudos, 12 artigos do tipo *philosophical paper* discutem como usar o OSLC para solucionar problemas, mas não apresentam sua implementação. Existem também diretrizes, opiniões e relatórios baseados em experiências profissionais na indústria, resultando em 5 artigos do tipo *experience report* e 1 artigo do tipo *opinion paper*, apontando que o OSLC é utilizado em contextos reais. A quantidade de artigos em cada um dos tipos de pesquisa é apresentada na Tabela 9.

Q4. Quais são as facetas da contribuição do estudo?

Identificar quais facetas de contribuição são mais exploradas em uma área de pesquisa é importante para encontrar tendências na área. Em relação à faceta contribuição

ID	Evaluation Research	Validation Research	Solution Proposal	Philosophical Paper	Experience Paper	Opinion Paper	ID	Evaluation Research	Validation Research	Solution Proposal	Philosophical Paper	Experience Paper	Opinion Paper
s1			X				s31			X			
s2			X				s32					X	
s3				X			s33				X		
s4				X			s34			X			
s5			X				s35			X			
s6			X				s36	X					
s7			X				s37			X			
s8			X				s38				X		
s9			X				s39	X					
s10		X					s40					X	
s11			X				s41			X			
s12			X		X		s42			X			
s13				X			s43			X			
s14			X				s44					X	
s15			X				s45				X		
s16			X				s46			X			
s17			X				s47			X			
s18			X				s48	X					
s19				X			s49			X			
s20	X						s50			X			
s21		X					s51				X		
s22			X				s52				X		X
s23			X				s53				X		
s24		X					s54			X			
s25			X				s55			X			
s26				X			s56			X			
s27			X				s57			X			
s28			X				s58					X	
s29	X						s59			X			
s30			X										

Tabela 8 – Estudos Distribuídos pelos Tipos de Pesquisa

fonte: O autor.

Tipo de Pesquisa	**Quantidade**
Evaluation Research	5
Validation Research	3
Solution Proposal	36
Philosophical Paper	12
Experience Paper	5
Opinion Paper	1

Tabela 9 – Quantidade de Estudos em Cada Tipo de Pesquisa

fonte: O autor.

dos estudos, foram estabelecidas 5 categorias, conforme pode ser visto na Tabela 10. Essas facetas são:

- Ferramenta: categoria composta por adaptadores, serviços web, software para automatizar atividades, etc.
- Modelo: categoria composta por representações gráficas de uma atividade ou componentes de software em uma especificação padrão.
- Processo: é um conjunto de atividades que conduz ao desenvolvimento de um software ou à execução do projeto.
- Método: é composto por abordagens que descrevem como usar uma tecnologia durante o ciclo de vida de desenvolvimento do software.
- Métrica: é um padrão qualitativo ou quantitativo usado para medir um sistema ou processo de software.

Os resultados mostram que a maioria das contribuições dos trabalhos são referentes a faceta de ferramentas. Esta categoria está relacionada à implementação de adaptadores de ferramentas OSLC, serviços web para compartilhamento de dados, ferramentas para visualizar dados de artefatos ou cadeias de ferramentas integradas com base no OSLC.

Em relação à categoria do modelo, trabalhos como (ZHANG; MøLLER-PEDERSEN, 2014), (GüRDüR et al., 2018), (ALVAREZ-RODRíGUEZ et al., 2018), (GALLINA; PADIRA; NYBERG, 2016), (MUSTAFA; LABICHE, 2017) abrange uma proposta para estender as especificações de domínios OSLC. Outro tópico representativo na categoria de modelo são abordagens para gerar especificações acoslc em ambientes baseados em MDE e *Model-based Systems Engineering* (MBSE).

Nos processos encontrados, o OSLC é usado para automatizar atividades que envolvem a rastreabilidade de artefatos, conforme mostrado em trabalhos como (BAUMGART; ELLEN, 2014), (REGAN et al., 2015), (REGAN et al., 2014).

Em outros trabalhos, são propostas metodologias ou novas abordagens para o uso de OSLC. Assim, foi possível identificar que as contribuições dos estudos estão voltadas para a resolução de problemas em um domínio específico, como uma atividade ou um tipo de indústria. É um indicativo da lacuna de pesquisa de abordagens para soluções globais sobre OSLC, em processos especialmente planejados para ES. A quantidade de artigos em cada uma das facetas é apresentada na Tabela 11.

4.4 Ameaças às Validade

Os trabalhos empíricos devem ser capazes de conter informações suficientes para que seja possível replicá-lo apenas com o acesso ao protocolo. Entretanto, durante sua

ID	Ferramenta	Modelo	Processo	Método	Métrica	ID	Ferramenta	Modelo	Processo	Método	Métrica
s1			x			s31	x				
s2	x					s32					
s3		x				s33				x	
s4				x		s34	x	x			
s5	x					s35	x			x	
s6	x	x		x		s36	x				
s7				x		s37		x			
s8	x					s38					
s9	x			x		s39			X		
s10		x				s40					
s11		x				s41	X				
s12	x	x				s42	X				
s13		x				s43		X			
s14	x			x		s44					
s15	x	x				s45				X	
s16		x				s46	X				
s17	x	x				s47	X				
s18	x					s48	X			X	
s19				x		s49	X			X	
s20		x				s50	X				
s21	x	x				s51		X			
s22				x		s52				X	
s23				x		s53				X	
s24		x				s54	X			X	
s25	x					s55	X			X	
s26				x		s56	X	X			
s27		x				s57	X	X			
s28				x		s58	X				
s29	x	x				s59			X		
s30	x			x							

Tabela 10 – Estudos Distribuídos pelas Facetas

fonte: O autor.

Faceta	**Quantidade**
Ferramenta	30
Modelo	20
Processo	3
Métrica	0
Método	20

Tabela 11 – Quantidade de Estudos em Cada Faceta de Contribuição

fonte: O autor.

execução existem alguns fatores que podem interferir nos resultados, sendo tratados como ameaças à validade. A seguir seguem algumas ameaças deste SMS:

Validade interna: Esse tipo de ameaça refere-se à validade da análise realizada, validando se as conclusões derivadas dos dados são válidas internamente. Nesse sentido, perguntas de pesquisa bem definidas possibilitaram concluir resultados de acordo com o tópico de pesquisa. Além disso, uma preocupação constante do estudo foi garantir que o processo de seleção de trabalhos ocorresse conforme os critérios de inclusão e exclusão definidos. Para esse fim, foi cuidadosamente investido um grande esforço para filtrar os estudos primários, com o auxílio de ferramentas para organização e filtros dos trabalhos utilizados pelos pesquisadores.

Validade externa: Diz respeito sobre o quão generalizadas são as descobertas do estudo de mapeamento. Nesse sentido, a representatividade estatística pode ser uma ameaça. Por considerar um número expressivo de artigos, essa ameaça é tratada pelo refinamento da *string* de busca por mais de um pesquisador, permitindo a inclusão de 59 estudos primários, os quais caracterizam um bom poder estatístico, e a adaptação do texto para cada uma das bases selecionadas.

Validade de construção: O trabalho foi executado com base em um protocolo bem definido e amplamente utilizado na área de Engenharia de Software. Para garantir a extração dos dados, o esquema de classificação foi revisado e discutido por um segundo pesquisador. Além disso, o trabalho conta com critérios de qualidade para identificar os trabalhos mais relevantes para o tema de pesquisa. Entretanto, as categorias escolhidas para o esquema de classificação podem ser consideradas genéricas demais. Para mitigar esse viés, as categorias foram escolhidas a partir da base de conhecimento do RUP.

Validade da conclusão: Uma ameaça a conclusão pode ocorrer na fase de classificação, no qual pode ser influenciada pelo viés do *know-how* dos autores. Para superar esse viés, foi utilizado um formulário de extração de dados que contém todos os dados necessários para responder as questões de pesquisa.

4.5 Lições do Capítulo

Este Capítulo apresentou a busca dos trabalhos relacionados por meio de um SMS. Para isso, seguiu-se um protocolo bem definido composto por quatro questões de pesquisa com o objetivo de identificar o estado da arte e prática do uso do OSLC. Assim, 59 artigos foram selecionados como relevantes para a pesquisa.

Os resultados mostram que as empresas que o adotam baseiam-se em ambientes de sistemas críticos e são consideradas de grande porte. Apesar disso, seu uso não cobre todo o ciclo de vida do software.

Ainda, a integração de aplicações com o OSLC baseiam-se principalmente em integrações ponto a ponto com adaptadores. Com base nisso, há interesse da indústria em metodologias, processos e tecnologias MDD para a geração de código-fonte de interfaces

adaptadoras. O Capítulo 5 apresenta um estudo exploratório de uma ferramenta MDD encontrada na literatura para o desenvolvimento de adaptadores OSLC.

5 DESENVOLVIMENTO DE ADAPTADORES OSLC

Este trabalho, de caráter exploratório, possui o objetivo de identificar os aspectos envolvidos na modelagem de integração por meio do Eclipse Lyo e geração e códigos com base em artefatos em conformidade com o padrão OSLC. Para isso, pretende-se argumentar sobre elementos necessários ao longo do processo de aprendizado para se usar a interface do ambiente Eclipse, identificando as atividades necessárias para implementar adaptadores e mapear as vantagens e desvantagens do uso do Eclipse Lyo.

O Eclipse Lyo é uma ferramenta com base em MDD encontrada nos trabalhos relacionados, abordado no Capítulo 4. É uma das tecnologias adotadas na indústria para criar interfaces de ferramentas, que visam a troca de dados interoperada por meio de adaptadores OSLC.

A implementação de um adaptador OSLC requer que muitas decisões sejam tomadas pelos desenvolvedores. Por exemplo: (i) identificar quais são os dados que devem ser providos/consumidos pelas ferramentas; (ii) quais especificações de domínios utilizar; (iii) qual o fluxo da troca de mensagens, dentre outras discutidas em (LEITNER; HERBST; MATHIJSSEN, 2016).

Nesse sentido, com o objetivo de explorar a ferramenta Eclipse Lyo, buscou-se desenvolver uma solução de integração OSLC construída na topologia ponto a ponto, por meio de adaptadores OSLC.

5.1 Caracterização do Cenário

Com o objetivo de explorar a ferramenta Eclipse Lyo, buscou-se analisar parte de um cenário da Diretoria de Tecnologia da Informação e Comunicação (DTIC), departamento de tecnologia da Universidade Federal do Pampa (Unipampa). Este cenário foi parcialmente implementado, resultando em algumas contribuições como segue: (i) identificada a carência no estado da arte de um material que sintetiza o organiza atividades para a geração de soluções de integração em OSLC, derivou-se dessa experiência um processo para desenvolvimento de integradores; (ii) identificação e relato das dificuldades que uma equipe em cenário real enfrentará ao utilizar o aparato ferramental explorado; e (iii) comprovação na prática de que o padrão OSLC e o Eclipse Lyo funcionam na integração dos artefatos do cenário explorado.

As ferramentas exploradas neste trabalho, e que compõem o cenário de integração, são mostradas na Tabela 12. Elas foram selecionadas com base no ecossistema de software da DTIC. Esse setor de desenvolvimento da Unipampa trabalha com softwares livres e de código aberto, selecionando em seu ambiente de produção ferramentas que suportam atividades de desenvolvimento, gestão e comunicação. Dentre o aparato ferramental utilizado, este estudo teve por objetivo testar OSLC e Eclipse Lyo em um conjunto de três ferramentas que melhor representam um cenário de Engenharia de Software Contínua para o DTIC. Então, para melhor explorar a heterogeneidade do cenário, explorou-se o

ID	Ferramenta	Contexto	URL
T01	Redmine	gerenciamento de projeto	http://www.redmine.org
T02	Libreoffice /Excel	Gerenciamento de requisitos	http://pt-br.libreoffice.org
T03	Testlink	Gerenciamento de testes	http://testlink.org
T04	Mantis	Gerenciamento de testes	https://www.mantisbt.org
T05	GLPI	Gerenciamento de serviços de TI	https://glpi-project.org/pt-br
T06	Subversion	Controle de versões	https://subversion.apache.org
T07	Eclipse	Ambiente de desenvolvimento	https://www.eclipse.org

Tabela 12 – Ferramentas levantadas para integração no contexto do DTIC - Unipampa

desenvolvimento de adaptadores numa integração de artefatos representados em arquivos, e dois adaptadores para integrações por meio de acesso à serviços: Libreoffice/Excel, Redmine e Testlink.

Em adição, a fim de explorar e ilustrar os benefícios da geração de código de adaptadores para ferramentas de diferentes domínios, bem como limitações no estado da arte, explorou-se os limites da ferramenta Eclipse Lyo. Esta foi selecionada principalmente porque é recomendada na página da OSLC.

5.2 Eclipse Lyo

O Eclipse Lyo é um projeto de código aberto que visa ajudar a comunidade interessada em integrações, principalmente na adoção das especificações OSLC em suas ferramentas e no desenvolvimento novas soluções compatíveis com o OSLC (EL-KHOURY, 2016). O projeto consiste nos seguintes componentes: OSLC4J SDK, que é um kit de desenvolvimento Java para interfaces de ferramentas provedoras e consumidoras OSLC e o Lyo Designer, que é um *workspace* do eclipse composto por uma DSL e geradores de códigos baseado em modelos, permitindo assim a representação gráfica dos modelos de integrações por meio de MDE.

Apesar de ser uma recomendação na página da OSLC, observou-se que o código gerado por meio do Eclipse Lyo é limitado. Ou seja, os geradores são mapeados apenas para a assinatura de um conjunto de métodos, no que é denominado por esqueleto de código. Assim, é necessária muita programação manual para que permitam a comunicação entre as ferramentas. Além disso, para acessar os dados armazenados internamente das ferramentas integradas e estabelecer as ligações, é necessário implementá-los manualmente, o que no caso do arquivo de planilha eletrônica significa o desenvolvimento de um *parser*, e das outras duas ferramentas, o desenvolvimento de interfaces de serviço, uma vez que não suportam o padrão OSLC.

Em relação à modelagem da solução de integração, identificou-se que existem três perspectivas para a representação de modelos no Lyo Designer: Perspectiva Especificação de Domínios, Perspectiva Cadeia de Ferramentas e Perspectiva Interface do Adaptador.

Cada perspectiva permite representar um tipo de diagrama como segue:

1. **Perspectiva Especificação de Domínios -** Trata sobre a representação dos domínios OSLC. É possível adicionar múltiplos domínios. Cada domínio é composto por *Resources* (artefatos) e *Resources Properties* (valores permitidos, cardinalidade e opcionalidade).

2. **Perspectiva Cadeia de Ferramentas -** Refere-se a estrutura da cadeia de ferramentas e artefatos que serão compartilhados entre as aplicações, definindo quais as ferramentas que serão consumidoras e provedoras.

3. **Perspectiva Interface do Adaptador -** Representa a estrutura dos adaptadores de cada ferramenta representada na perspectiva Cadeia de Ferramentas. Essa estrutura segue a especificação principal do OSLC, sendo a base para a geração de códigos.

As representações devem ser elaboradas de acordo com as decisões de projeto. Após a execução dessas três visões de modelagem, é possível gerar os códigos dos adaptadores. Por fim, para executar o adaptador é necessário importar os códigos para um projeto Maven Web.

Inicialmente, para aprender o uso do Eclipse Lyo, partiu-se para estudos das informações disponíveis em fóruns e listas de discussões. O material é escasso e não adequado para iniciantes. Observou-se de um modo exploratório que, para se desenvolver um adaptador OSLC no Eclipse Lyo, um conjunto de passos é necessário como segue: (i) Criar no ambiente Eclipse um projeto *Modelling Project*; (ii) Representar graficamente os modelos da cadeia de ferramentas OSLC; (iii) Validar se a representação dos modelos está de acordo com a regras do metamodelo; (iv) Gerar os códigos de forma automática; (v) Implementar manualmente o código necessário para acessar dados de outra ferramenta; (vi) Adicionar o código gerado em um projeto Maven Web no ambiente Eclipse; (vii) Executar o adaptador.

Após realizar as três perspectivas de modelagem, utilizou-se a abordagem disponível, baseadas em MDD, para gerar código-fonte para adaptadores OSLC. Portanto, executou-se a geração automatizada linear, sem intervenção do projetista da solução de integração, de um modelo independente para uma representação específica da plataforma.

5.3 Tutorial para Desenvolvimento da Integração

Este tipo de estudo exploratório pode e deve ser replicado em ambientes de desenvolvimento reais. Para tal, é essencial a definição de uma metodologia que sintetiza toda a bagagem adquirida ao longo da execução da integração em vistas à facilitar a implementação da Engenharia de Software Contínua. Por um motivo de fomentar a transferência

de conhecimento tácito em explícito, derivou-se um conjunto de atividades à ser desempenhada pelos engenheiros de software.

Assim, uma vez que os a documentação do Eclipse Lyo, e que uma contribuição mais metodológica é uma limitação no estado da arte do tema, como identificado nas publicações associadas com OSLC relatadas no SMS, estabeleceu-se um processo para o desenvolvimento de adaptadores OSLC no Eclipse Lyo, como ilustrado na Figura 13. Para seu início, é necessário que os desenvolvedores conheçam o escopo da cadeia de ferramentas, definindo quais delas vão compor a solução de integração.

Parte-se, portanto, do ponto onde o conjunto de ferramentas mostrados na Tabela 12 está bem definido pela organização/fábrica de software. Também é importante que ela tenha traçado seu objetivos em termos de automação dos processos de desenvolvimento. Também é fundamental que seus times utilizem estas ferramentas em seu quotidiano, para que não exista uma curva de aprendizado que não seja estritamente relacionado ao desenvolvimento da solução de integração.

Também é importante ressaltar que o foco do processo mostrado na Figura 13 é exclusivamente destinado para o perfil integrador. Ele viabilizará a colaboração em *toolchain* por meio de adaptadores que operem conforme o padrão OSLC, estando portanto focado na integração.

Com esse cenário definido, o próximo passo é estabelecer os relacionamentos entre as ferramentas. Nesse subprocesso/atividade, são definidos o fluxo da troca de dados ao longo da cadeia de ferramentas. A integração de ferramentas com OSLC é estabelecida por meio da topologia ponto a ponto. E dessa forma, cada uma das ferramentas possuem uma ou mais ligações com as demais. Essas conexões representam o fluxo da toca de dados.

Nesse sentido, em uma solução OSLC, as ferramentas podem ser classificadas como ferramentas provedoras e/ou consumidoras. Relacionado a isso, é necessário identificar quais os artefatos serão compartilhados entre as ferramentas. Por exemplo, uma ferramenta de gerenciamento de requisitos pode consumir dados de uma ferramenta de testes, enquanto uma ferramenta de análise estática de código-fonte pode prover relatórios e consumir requisitos ao mesmo tempo. Assim, não é obrigatório que todos os artefatos de uma ferramenta sejam compartilhados via OSLC, podendo o engenheiro de software ligar apenas dados chaves.

Identificados os artefatos que serão compartilhados, ainda resta selecionar quais os atributos serão representados e compartilhados via OSLC. Essa atividade geralmente é realizada com base na interface de usuário da ferramenta. Por isso, o filtro das propriedades essenciais é importante para encadear uma ferramenta no restante do processo. Por exemplo, em um cenário talvez não seja interessante representar a prioridade de um caso de teste, sendo esse filtrado dos dados que serão integrados com uma ferramenta de gerenciamento de projetos, por exemplo.

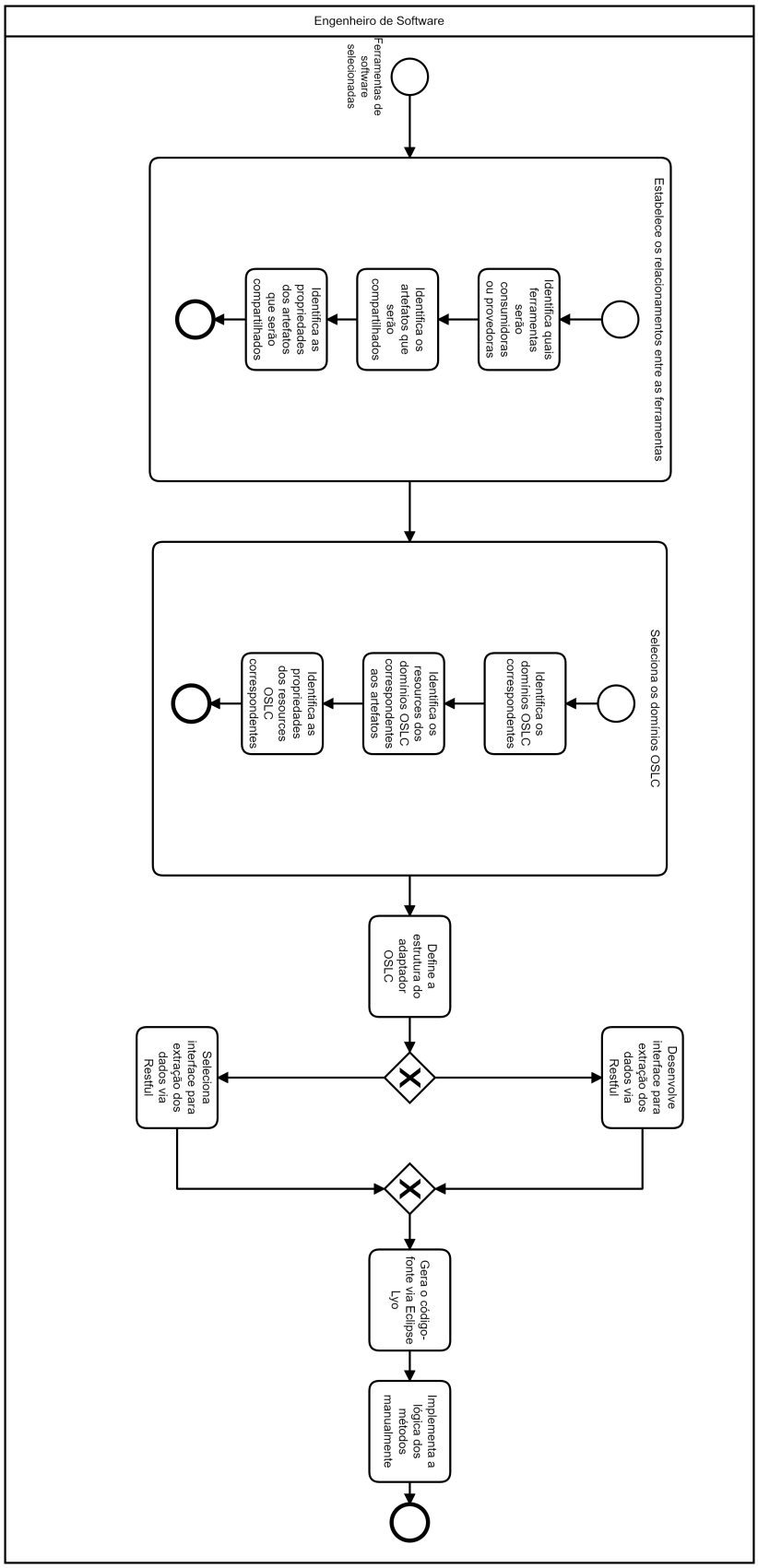


Figura 13 – Processo para o desenvolvimento de elementos para a integração de ferramentas em OSLC e Eclipse Lyo

Fonte: O autor.

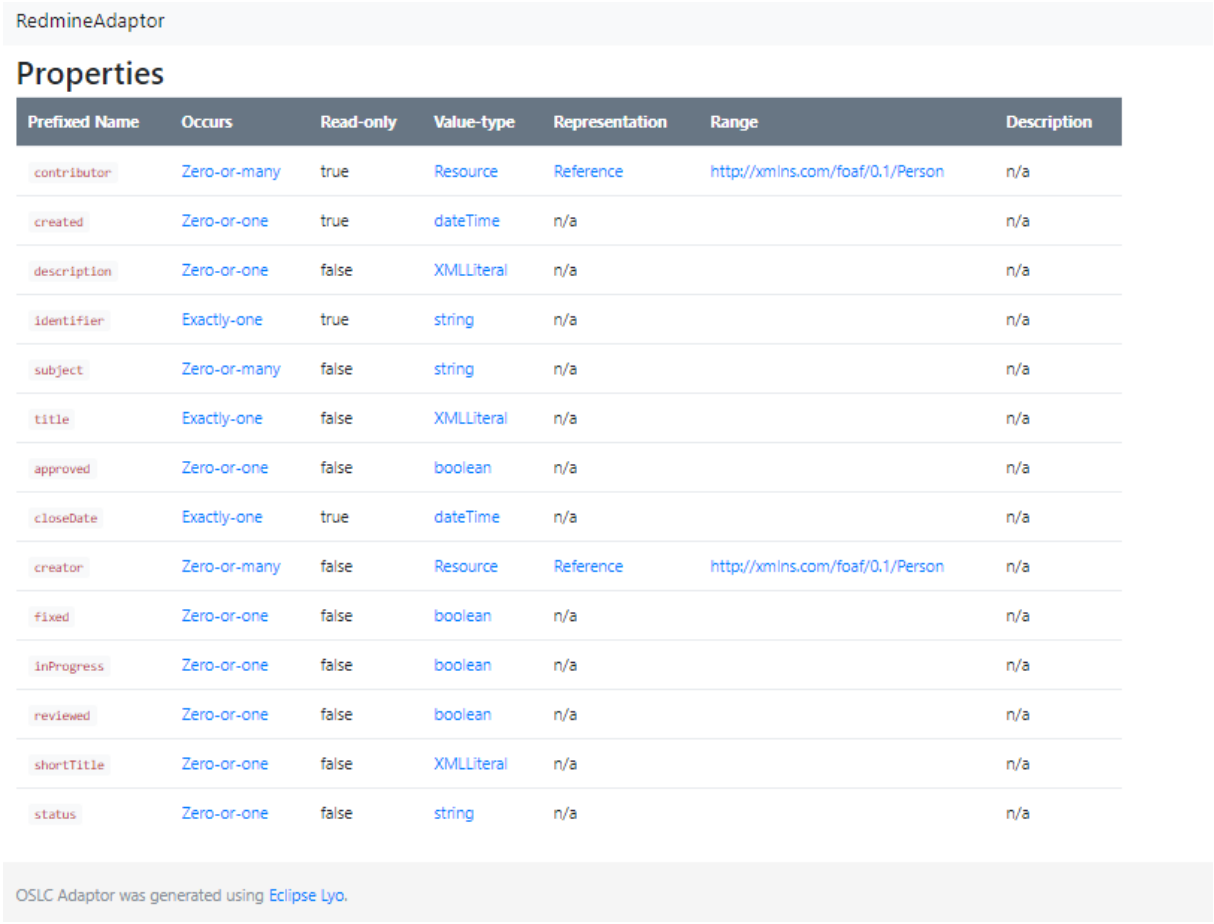
Além dos relacionamentos em nível de ferramenta, é necessário estabelecer quais os artefatos devem ser relacionados por meio dos dados ligados. Este relacionamento acontece por meio de propriedades específicas dos domínios OSLC. Por exemplo, o domínio de Gerenciamento de Qualidade possui a propriedade *validatesRequirement*, que mantém a informação sobre quais são os requisitos que um determinado caso de teste se aplica. Com base nisso, é possível estabelecer a rastreabilidade entre artefatos que são representados no formato RDF/XML, ligando dados chaves por meio de um adaptador. Ou seja, o RDF é o resultado da ligação de uma entrada e de uma saída trocadas entre duas ferramentas.

De modo à atender diversos tipos de ferramentas adotadas na produção de software, o consórcio de empresas e acadêmicos que mantém o OSLC propuseram uma série de domínios OSLC. Esses domínios significam que, na teoria, quaisquer ferramentas que pertençam a uma fase do ciclo de vida do software (*e.g.* requisitos, testes), mantém artefatos que podem ser representados por essas especificações. Essas informações sobre domínios não estão disponíveis no Eclipse Lyo, sendo necessário acessar as várias páginas *webs* correspondentes aos domínios no site do OSLC. Com base nisso, para escolher os domínios OSLC, deve-se olhar para os domínios das ferramentas selecionadas. Isso parece uma atividade trivial, mas é dificultada porque as ferramentas não são ilhas. Cada ferramenta pode ofertar um conjunto de artefatos que se sobrepõem, o que torna o processo de identificação dos elementos de interesse para o domínio algo bastante exploratório e suscetivo ao erro da interpretação do engenheiro de software.

Já quanto aos domínios disponíveis no site da OSLC, estes são definidos como descrição de atributos. Ou seja, não há uma informação preparada para o ensino de OSLC, apenas especificações técnicas das interfaces de serviços operadas por meio de RDF. Cada um dos domínios possui uma série de *resources* que são representações dos artefatos, com atributos que podem ser utilizados e sua definição de tipo de dados. Por exemplo, o domínio de Gerenciamento de Testes possui os *resources* de Caso de Testes, Plano de Testes, etc. Além disso, esses *resources* possuem propriedades/atributos correspondentes aos artefatos, como ilustra a Figura 14 pela visualização de um *resource* OSLC. O exemplo demonstra dados de recursos que estão disponíveis por uma ferramenta no domínio de Gerenciamento de Testes.

Além dos domínios principais, existem os domínios auxiliares que servem para representar outros tipos de dados. Por exemplo, os domínios FOAF e dcterms são padrões para representar dados pessoais na web. No cenário investigado, assuma que se está integrando dados no contexto do TestLink com o RedMine. Por exemplo, para representar o criador de um plano de testes, o domínio de gerenciamento de qualidade não possui um *resources* “pessoa” associado com o plano.

Outra atividade essencial é definir a estrutura dos adaptadores OSLC. Para isso, é necessário seguir a especificação principal *(Core)* do OSLC, a qual é definida pela *Asset Management Specification (AMS)*. Esta especificação define os elementos essenciais de to-



RedmineAdaptor

Properties

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
contributor	Zero-or-many	true	Resource	Reference	http://xmlns.com/foaf/0.1/Person	n/a
created	Zero-or-one	true	dateTime	n/a		n/a
description	Zero-or-one	false	XMLLiteral	n/a		n/a
identifier	Exactly-one	true	string	n/a		n/a
subject	Zero-or-many	false	string	n/a		n/a
title	Exactly-one	false	XMLLiteral	n/a		n/a
approved	Zero-or-one	false	boolean	n/a		n/a
closeDate	Exactly-one	true	dateTime	n/a		n/a
creator	Zero-or-many	false	Resource	Reference	http://xmlns.com/foaf/0.1/Person	n/a
fixed	Zero-or-one	false	boolean	n/a		n/a
inProgress	Zero-or-one	false	boolean	n/a		n/a
reviewed	Zero-or-one	false	boolean	n/a		n/a
shortTitle	Zero-or-one	false	XMLLiteral	n/a		n/a
status	Zero-or-one	false	string	n/a		n/a

OSLC Adaptor was generated using Eclipse Lyo.

Figura 14 – *Resource* OSLC correspondente ao Artefato do Tipo Tarefa

fonte: O autor.

dos os recursos, incluindo elementos utilizados para federação de dados inter-repositórios, que também serve para estabelecer dados ligados nos chamados de *cloud-of-clouds* implementados no padrão OSLC. Nessa especificação, todos os artefatos são compartilhados via serviços interoperáveis numa interface comum. Para isso, cada um deles possui um URI, que localiza o recurso. Com isso, por meio de operações Restful é possível acessá-los e manipulá-los por meio de funções como *Get* e *Post*.

Com base nesse cenário de dados ligados de modo distribuído, a fim de organizar o projeto e permitir o acesso à esses serviços, deve-se também definir a estrutura desses adaptadores por meio de catálogos. Esses catálogos disponibilizam URIs para os provedores de serviços e seus serviços, habilitando assim a localização de serviços entre repositórios (*inter-cloud*). Os provedores de serviços hospedam estes catálogos e permitem organizar os serviços de forma que seja atrativo aos interesses do projeto, cada um sendo regido por suas restrições de acesso.

Assim como nas ferramentas, os artefatos devem estar organizados no adaptador de uma forma que seja possível localizá-los. Para isso, é trabalho dos desenvolvedores projetar a melhor forma para organizar os artefatos mantidos nas ferramentas. Por exemplo, é

possível filtrar os dados sobre defeitos de uma ferramenta *bugtracker* através dos projetos. Ou ainda, filtrar as tarefas em uma ferramenta de gestão de projetos pelos times. Nesse sentido, os provedores de serviços seriam os projetos e os times. Portanto, por meio dos adaptadores destas ferramentas, o catálogo de provedores de serviços pode elencar todos os projetos do *bugtracker* ou times da ferramenta de gestão de projetos.

Com essas definições, mais as atividades planejadas para o desenvolvimento dos adaptadores, do desenho da integração, da definição das interfaces dos serviços das ferramentas, então pode-se partir para a geração do código dos adaptadores por meio do Eclipse Lyo. Quanto ao desenvolvimento das interfaces de serviço, para que seja possível manipular os dados, primeiro deve-se buscar e estudar uma API RESTful para cada uma das ferramentas envolvidas. Estas soluções podem estar prontas, mas identificou-se que a sua disponibilização de forma gratuita é bastante rara. Então, é provável que as interfaces de serviço precisem ser desenvolvidas para cada contexto organizacional. Portanto, caso não seja possível localizar as interfaces, deve-se criar sua própria API para a extração dos dados.

Ao final, após validar o projeto, gera-se os códigos fonte. No entanto, percebe-se que apenas os esqueletos dos códigos-fonte são gerados pelo aparato ferramental do Eclipse Lyo, sendo necessário implementar a lógica de cada um dos adaptadores e das interfaces de serviço. Para isso, são implementadas as funções REST com base na API selecionada para acesso à ferramenta integrada. Isso requer do engenheiro de software uma longa curva de aprendizado, principalmente para se entender o mecanismo interno de muitas ferramentas para a representação dos artefatos.

5.4 Demonstração Conceitual

Esta seção detalha a implementação de adaptadores OSLC por meio da geração de código-fonte com base em modelos no Eclipse Lyo.

Atividade 1 - Estabelece relacionamentos entre as ferramentas.

Atividade 1.1 - Identificar quais ferramentas serão consumidoras ou provedoras. **Entradas:** Ferramentas da *toolchain*. **Saídas:** Ferramentas provedoras e consumidoras identificadas. **Exemplificação:** A Figura 15 representa o resultado dessa atividade. Por exemplo, com o objetivo de ligar os requisitos com tarefas e casos de teste, foi projetado que a ferramenta de gerenciamento de requisitos seria consumidora de tarefas e casos de testes. Enquanto isso, para que esse relacionamento seja estabelecido, as demais ferramentas foram modeladas como provedoras.

Atividade 1.2 - Identificar os artefatos que serão compartilhados. **Entradas:** Ferramentas provedoras e consumidoras identificadas. **Saídas:** Artefatos que serão compartilhados identificados. **Exemplificação:** A Figura 15 também contém os artefatos que serão compartilhados. Essa atividade está diretamente relacionada à atividade 1.1. Entretanto, durante esta atividade, os desenvolvedores também devem pensar em outros

artefatos que podem servir para organizar esses artefatos selecionados. Por exemplo, os casos de testes podem ser organizados pelos planos de teste ou por meio dos projetos.

Atividade 1.3 - Identificar as propriedades dos artefatos que serão compartilhados **Entradas:** Artefatos que serão compartilhados identificados. **Saídas:** Atributos dos artefatos identificados. **Exemplificação:** Os atributos de cada artefato que serão compartilhados e visualizados nos adaptadores OSLC devem suprir as necessidades do projeto. Para isso, os desenvolvedores devem selecionar atributos que agreguem algum tipo de benefício as partes interessadas. Por exemplo, é interessante visualizar as datas de criação e de atualização do artefato, como mostra a Figura 14.

Atividade 2: Selecionar os domínios OSLC

Atividade 2.1: Identificar os domínios OSLC correspondentes **Entradas:** Ferramentas **Saídas:** Domínios OSLC identificados. **Exemplificação:** Os domínios OSLC são propostos para representar artefatos de ferramentas utilizadas nas atividades do desenvolvimento como elicitar requisitos, representar modelos, escrever casos de testes, etc. Nesse sentido, devem se identificar os domínios das ferramentas selecionadas com os domínios disponíveis. Por exemplo, esse trabalho abrange 3 domínios: Gerenciamento de Requisitos, Gerenciamento de Projeto e Gerenciamento de Qualidade.

Atividade 2.2: Identificar os *resources* dos domínios OSLC correspondentes aos artefatos **Entradas:** Domínios OSLC identificados. **Saídas:** *Resources* OSLC identificados. **Exemplificação:** Cada domínio OSLC possui diversos *resources* que servem para representar os artefatos de qualquer ferramenta desses domínios. Por exemplo, o domínio de gerenciamento de qualidade possui os *resources* para planos de teste, casos de teste, etc.

Atividade 2.3 Identificar as propriedades dos resources OSLC correspondentes **Entradas:** *Resources* OSLC identificados. **Saídas:** Atributos dos *resources* OSLC identificados. **Exemplificação:** Os *resources* possuem uma série de atributos para cobrirem o máximo de ferramentas possível do mesmo domínio. Entretanto, não são necessárias representar todos os atributos de um *resource* na solução OSLC. Os atributos selecionados devem ser equivalentes aos atributos considerados relevantes para o contexto, escolhidos na atividade 1.3.

Atividade 3: Definir a estrutura do adaptador OSLC **Entradas:** Relacionamentos entre as ferramentas definidos. **Saídas:** Estrutura do adaptador. **Exemplificação:** Essa atividade corresponde as funcionalidades que serão disponibilizadas nos serviços web. Por exemplo, a Figura 17 mostra a estrutura do adaptador. Nesse cenário, há um fluxo que segue o catálogo de provedores de serviços (todos os projetos), provedores de serviços (um projeto específico) e serviços (as tarefas). Acessando cada tarefa, é possível selecionar, criar, listar todas as tarefas e visualizar informações do *resource* do serviço.

Atividade 4: Desenvolver ou Selecionar uma interface para a extração dos dados via Restful **Entradas:** Ferramentas identificadas. **Saídas:** APIs desenvolvidas/selecio-

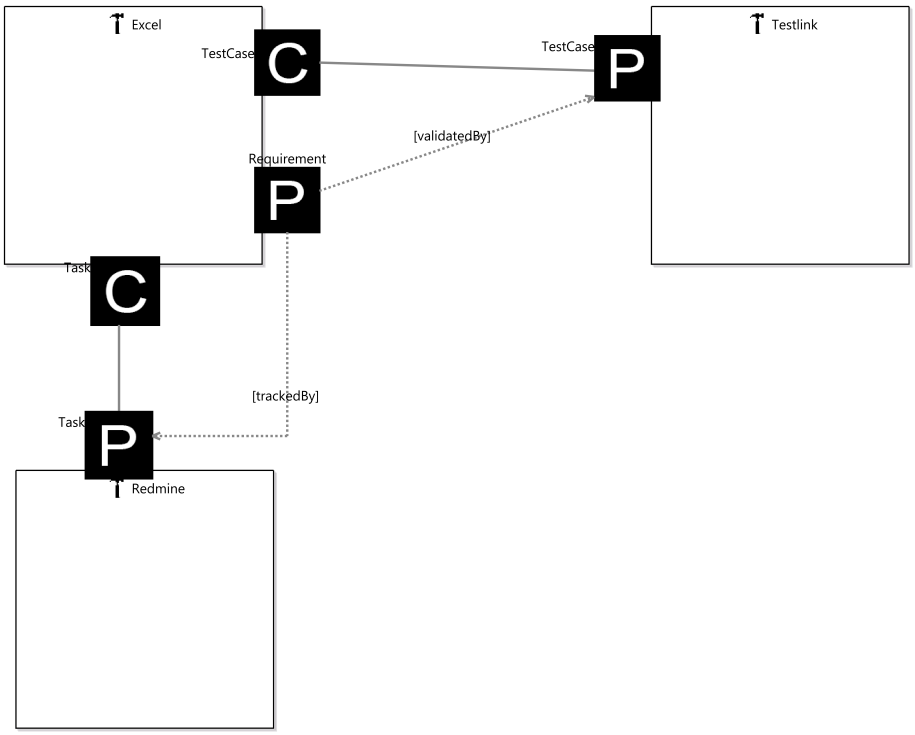


Figura 15 – Perspectiva Cadeia de Ferramentas

fonte: O autor.

nadas. **Exemplificação:** Essa atividade não é executada dentro do Eclipse Lyo. Apesar disso, é importante para que a integração seja realizada por meio do OSLC, uma API Restful para manipular os dados. Com isso, os desenvolvedores podem procurar em fóruns e sites oficiais das ferramentas em busca de uma API. Caso não seja encontrada, será necessário implementar do zero uma nova interface.

Atividade 5: Gerar o código-fonte via Eclipse Lyo **Entradas:** Modelos **Saídas:** Código-fonte **Exemplificação:** Após realizar a modelagem nas perspectivas do Eclipse Lyo com base nas atividades anteriores, é necessário gerar o código-fonte.

Atividade 6:Implementar a lógica dos métodos manualmente **Entradas:** Código-fonte e APIs **Saídas:** Adaptador OSLC **Exemplificação:** Essa atividade inclui importar o projeto para o ambiente Eclipse em um projeto Maven e utilizando os métodos disponíveis das APIS selecionadas na Atividade 4, desenvolver a lógica.

A Figura 15 mostra a cadeia de ferramentas do nosso estudo exploratório, composta por três ferramentas de diferentes domínios, sendo elas Excel/Libreoffice para o gerenciamento de requisitos, Redmine para o gerenciamento de projetos e Testlink para gerenciamento de testes.

A ferramenta Excel/Libreoffice mantém requisitos de software e deseja-se ligar por meio das propriedades dos dados ligados do OSLC às tarefas gerenciadas pelo Redmine e aos casos de testes do Testlink. Com base nisso, o adaptador OSLC do Excel/Libreoffice possui duas interfaces consumidoras para tarefas e casos de testes respectivamente e uma

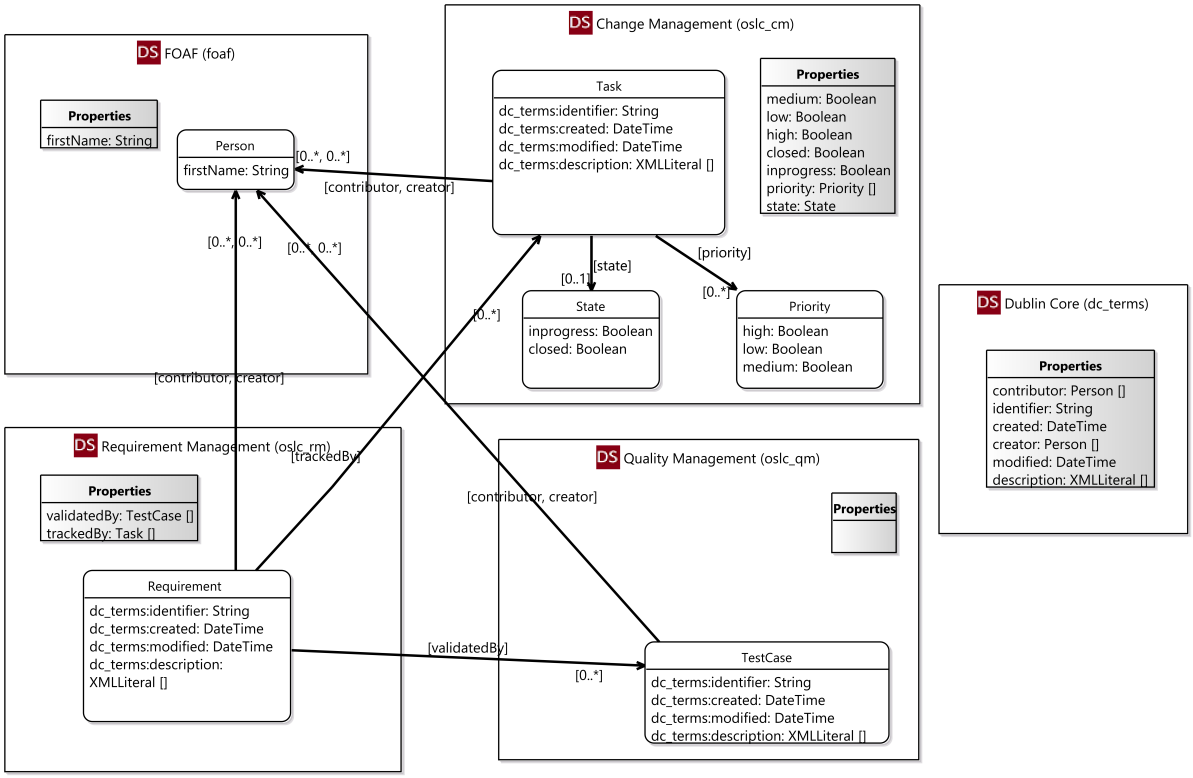


Figura 16 – Representação das Especificações do Domínio

fonte: O autor.

interface que provê requisitos de software. Além disso, as ferramentas Redmine e Testlink possuem interfaces provedoras para o compartilhamento de dados.

Embora nosso cenário tenha definido a estrutura da cadeia de ferramentas do nosso estudo, o primeiro passo para gerar os códigos-fonte dos adaptadores é a representação gráfica dos domínios OSLC. Como mostra a Figura 16, exploramos três domínios OSLC: Requirements Management (RM)[1], Change Management (CM)[2] e Quality Management (QM)[3].

Cada um desses domínios possui propriedades e *resources* para a representação dos artefatos mantidos pelas ferramentas e seus relacionamentos. Por exemplo, o domínio RM possui o *resource* Requirement com as propriedades que o descrevem como identificador, data de criação e modificação. Além disso, nessa representação são estabelecidos os relacionamentos entre os artefatos a partir de propriedades como a *validateBy* em que é possível ligar um caso de teste que valida determinado requisito. Além dos domínios OSLC, também foram explorados outros domínios auxiliares, os quais têm objetivo de

[1] http://docs.oasis-open.org/oslc-domains/oslc-rm/v2.1/oslc-rm-v2.1-part2-requirements-management-vocab.html

[2] http://docs.oasis-open.org/oslc-domains/cm/v3.0/cs02/part2-change-mgt-vocab/cm-v3.0-cs02-part2-change-mgt-vocab.html

[3] https://docs.oasis-open-projects.org/oslc-op/qm/v2.1/psd02/quality-management-vocab.html

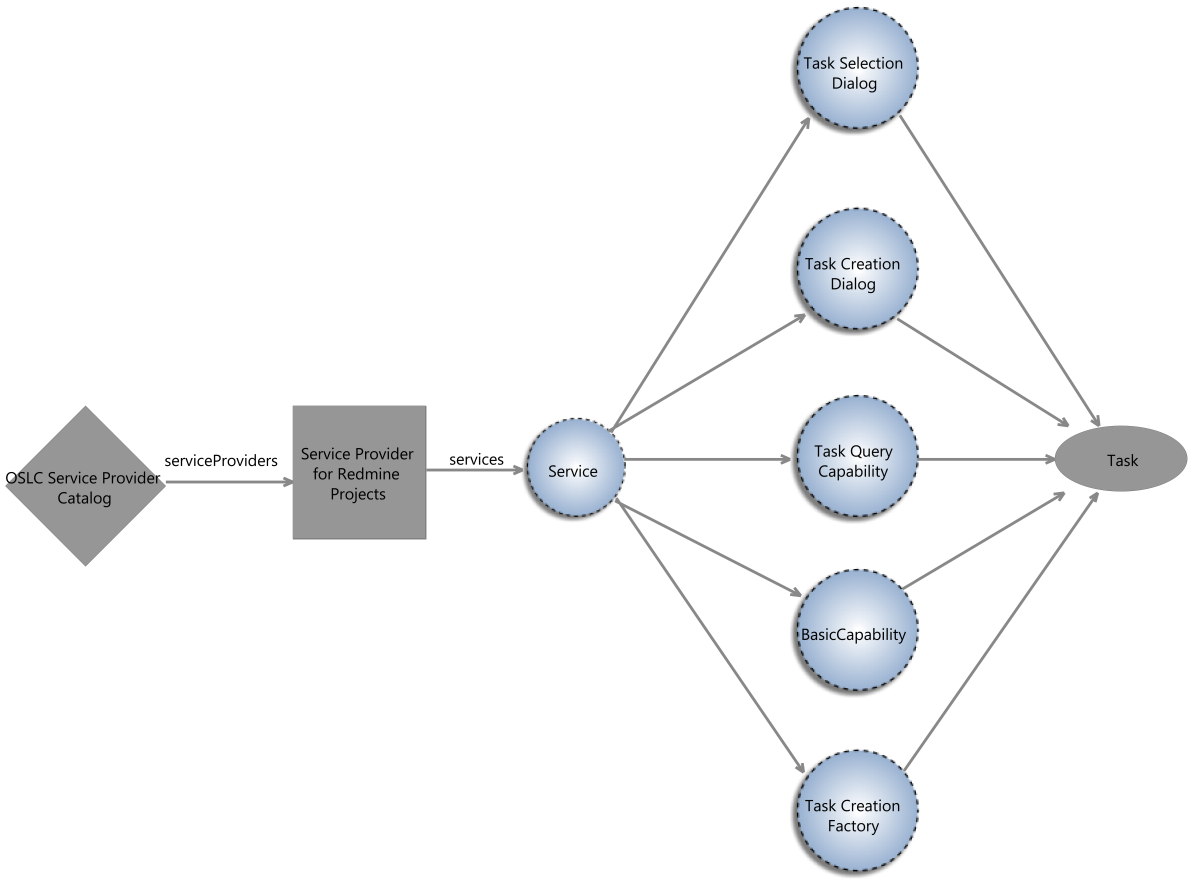


Figura 17 – Perspectiva Interface do Adaptador

fonte: O autor.

padronizar informações da web, como o FOAF [4] e Dublin Core [5].

A partir de serviços OSLC, é possível recuperar, editar, atualizar e excluir artefatos de software gerados pela cadeia de ferramentas. Essas funcionalidades são modeladas na perspectiva do adaptador. A Figura 17 mostra a interface do adaptador da ferramenta Redmine. A estrutura desse adaptador segue as especificações da Especificação principal do OSLC. Em adaptadores OSLC, todos os serviços são acessados por meio de *Uniformd Resource Identifier* (URIs). Ao acessar o adaptador é disponibilizado um catálogo de serviços (*Service Provider Catalog*) que mantém URIs para todos os provedores de serviços (*Service Providers*), os quais representam os projetos cadastrados no Redmine. Cada provedor de serviço disponibiliza URIs para acessar o serviços (*Services*), que nesse cenário são as tarefas de cada um dos projetos. Ao final do fluxo, é possível acessar cada uma das tarefas e manipulá-las por meio das funções RESTful as quais foram modeladas na perspectiva da interface do adaptador.

O Eclipse Lyo gera apenas os esqueletos de classes Java, sendo necessário implementar manualmente a lógica dos adaptadores. Para que isso seja possível, as ferramen-

[4] http://xmlns.com/foaf/spec/

[5] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

tas devem possuir alguma interface que possibilite manipular seus dados, como uma API RESTful. Nesse exemplo, foram utilizadas três APIs RESTful, uma para cada ferramenta envolvida no processo. Graças ao componente OSLC4J do Eclipse Lyo, todos os objetos tornam-se possível de serem representados nos formatos RDF/XML e JSON, possibilitando a troca de dados entre as ferramentas após a transformação dos seus dados em objetos Java para o padrão OSLC.

O código-fonte mostrado na Figura 18 ilustra a implementação do método para recuperar os projetos da ferramenta Redmine que servem como provedores de serviço nessa demonstração conceitual. Outro exemplo, é o método para recuperar tarefas do Redmine, exemplificado no código-fonte da Figura 19. Ambos os métodos foram desenvolvidos com uma API Restful e possuem a manipulação de uma lista de objetos que são alimentadas com os artefatos das ferramentas integradas em comum. O Eclipse Lyo tem um papel importante por transformar esses objetos nas especificações OSLC, os quais são acessados pelos serviços web por meio de uma URI.

```
public static List<ServiceProviderInfo> getServiceProviderInfos(HttpServletRequest httpServletRequest)  {
    String uri = "http://192.168.1.15/redmine/";
    String apiAccessKey = "5ee56404076e16bccad438b62a3e5bccd30bae53";
    List<Project> projects;
    List<ServiceProviderInfo> serviceProviderInfos = new ArrayList<ServiceProviderInfo>();

    com.taskadapter.redmineapi.RedmineManager mgr = RedmineManagerFactory.createWithApiKey(uri, apiAccessKey);
    mgr.setObjectsPerPage(100);
    try {
         projects = mgr.getProjectManager().getProjects();

         for (Project project : projects) {
            ServiceProviderInfo info = new ServiceProviderInfo();
            info.setName(project.getName());
            info.setServiceProviderId(project.getIdentifier());
            serviceProviderInfos.add(info);
        }

    }catch(Exception e) {
        System.out.println(e.getMessage());
    }
    return serviceProviderInfos;
}
```

Figura 18 – Exemplo de Código-Fonte de Um Adaptador OSLC do Catálogo de Provedores de Serviços

Fonte: O autor.

As Figuras 20 e 21 mostram interfaces gráficas do adaptador para a ferramenta Redmine. É possível ver o URI do provedor de catálogo para todos os projetos cadastros na ferramenta. Ao acessar qualquer um dos projetos as partes interessadas terão acesso as tarefas cadastradas naquele projeto com seus respectivos dados. As tarefas são representadas em arquivos RDF seguindo a estrutura da especificação OSLC, como mostra a Figura 22. Essa representação permite a troca de dados e assim, todas as ferramentas que possuem suporte ao OSLC, poderão consumi-las futuramente.

A Figura 23 mostra os elementos que foram modelados na perspectiva da interface

```java
public static List<Task> queryTasks(HttpServletRequest httpServletRequest, final String serviceProviderId, String where, int page, int limit)
{

    List<Task> resources = new ArrayList<Task>();

    String uri = "http://192.168.1.15/redmine/";
    String apiAccessKey = "5ee56404076e16bccad438b62a3e5bccd30bae53";

    final Map<String, String> params = new HashMap<String, String>();
    params.put("project_id", serviceProviderId);
    com.taskadapter.redmineapi.RedmineManager mgr = RedmineManagerFactory.createWithApiKey(uri, apiAccessKey);
    mgr.setObjectsPerPage(100);
    try {
        List<Issue> issues = mgr.getIssueManager().getIssues(params).getResults();

        for (Issue issue : issues) {

            Task t = new Task();

            t.setTitle(issue.getDescription());
            t.setIdentifier(issue.getId().toString());
            t.setCloseDate(issue.getClosedOn());
            t.setCreated(issue.getCreatedOn());
            t.setAbout(RedmineAdaptorResourcesFactory.constructURIForTask(serviceProviderId, t.getIdentifier()));

            resources.add(t);
        }
    }catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return resources;
}
```

Figura 19 – Exemplo de Código-Fonte de Um Adaptador OSLC para Recuperar Artefatos

Fonte: O autor.

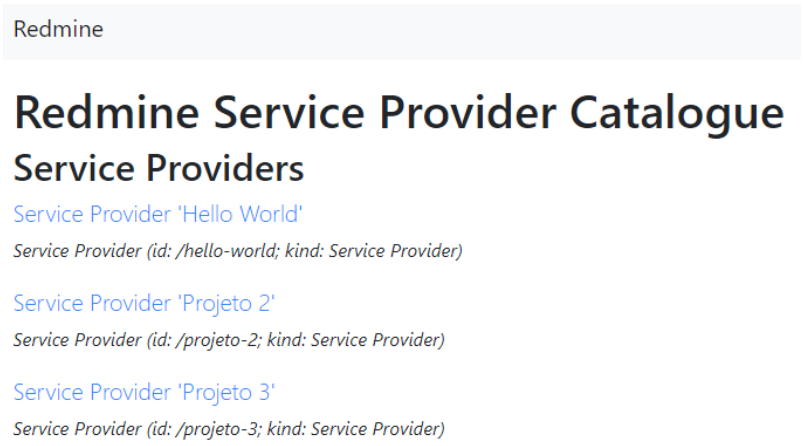


Figura 20 – Interface Gráfica do Catálogo de Provedores no Adaptador OSLC

Fonte: O autor.

do adaptador no Lyo e as classes *JavaServer Pages* (JSP) que são geradas correspondentes. Essas páginas possuem formulários básicos sem nenhum tipo de customização. Apesar disso, percebe-se a ligação direta entre a representação em modelos e o código-fonte gerado.

Além das classes para interface gráfica, a representação em modelos possibilita a geração de códigos para manipular os artefatos das ferramentas como objetos java. A Figura 24 mostra uma interface java para o domínio OSLC *Change Management* gerado a partir da representação gráfica no Lyo. Para cada *resource* modelado são geradas constantes que são utilizados para criar os URIs.

A Figura 25 mostra a interface gráfica que possui a opção de selecionar requisitos e os códigos correspondentes. O requisito é filtrado pelo identificador no qual serve como

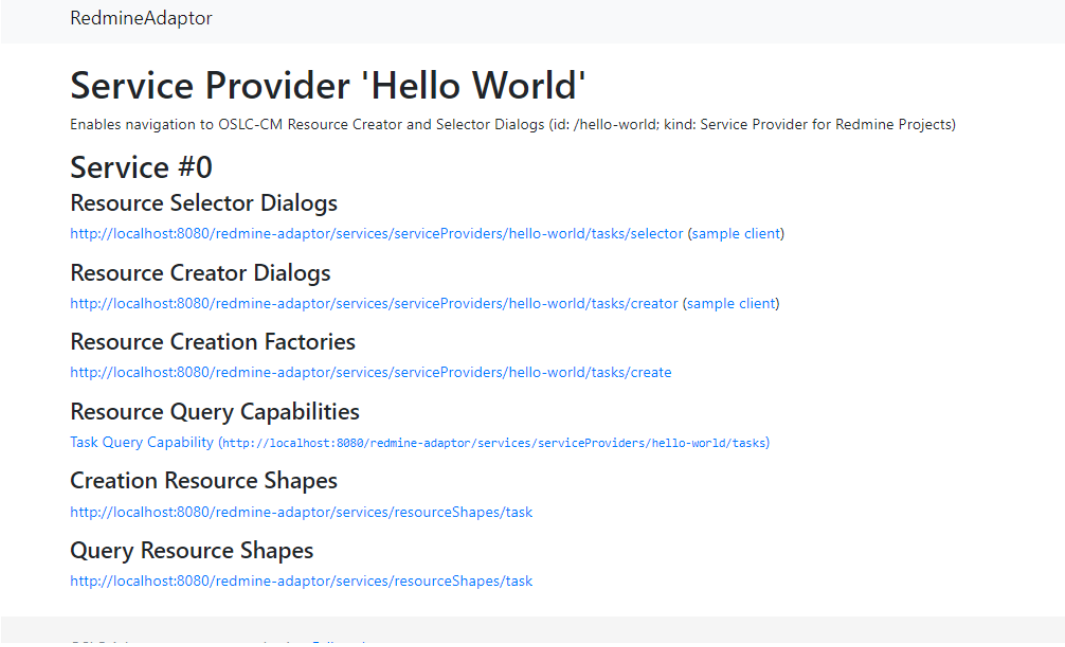


Figura 21 – Interface Gráfica dos Provedores de Serviços no Adaptador OSLC

Fonte: O autor.

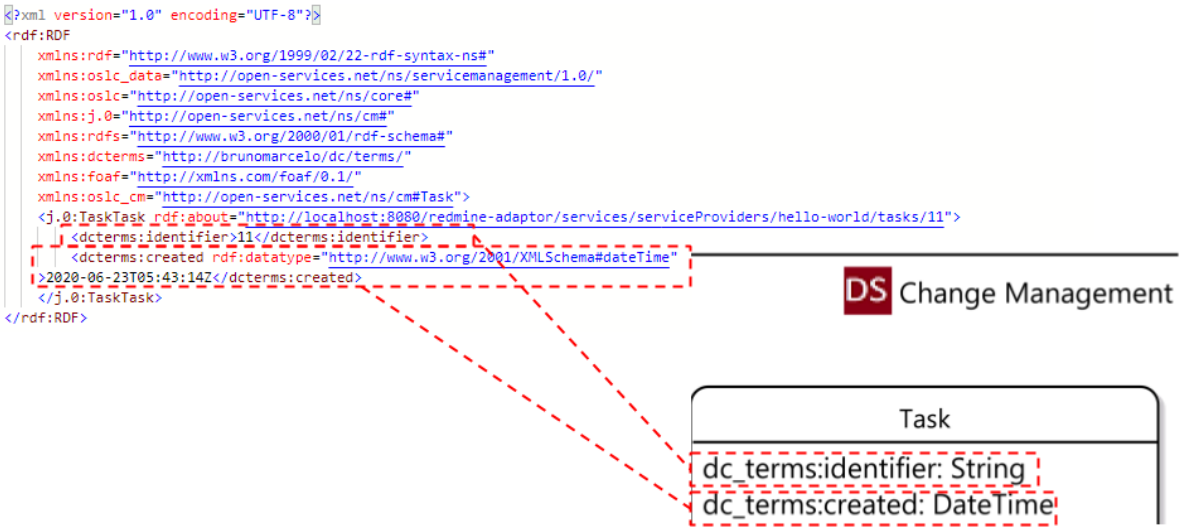


Figura 22 – Representação das Tarefas do Redmine nas Especificações OSLC

Fonte: O autor.

Figura 23 – Relação Entre Representação da Interface do Adaptador e Arquivos JSP Gerados

Fonte: O autor.

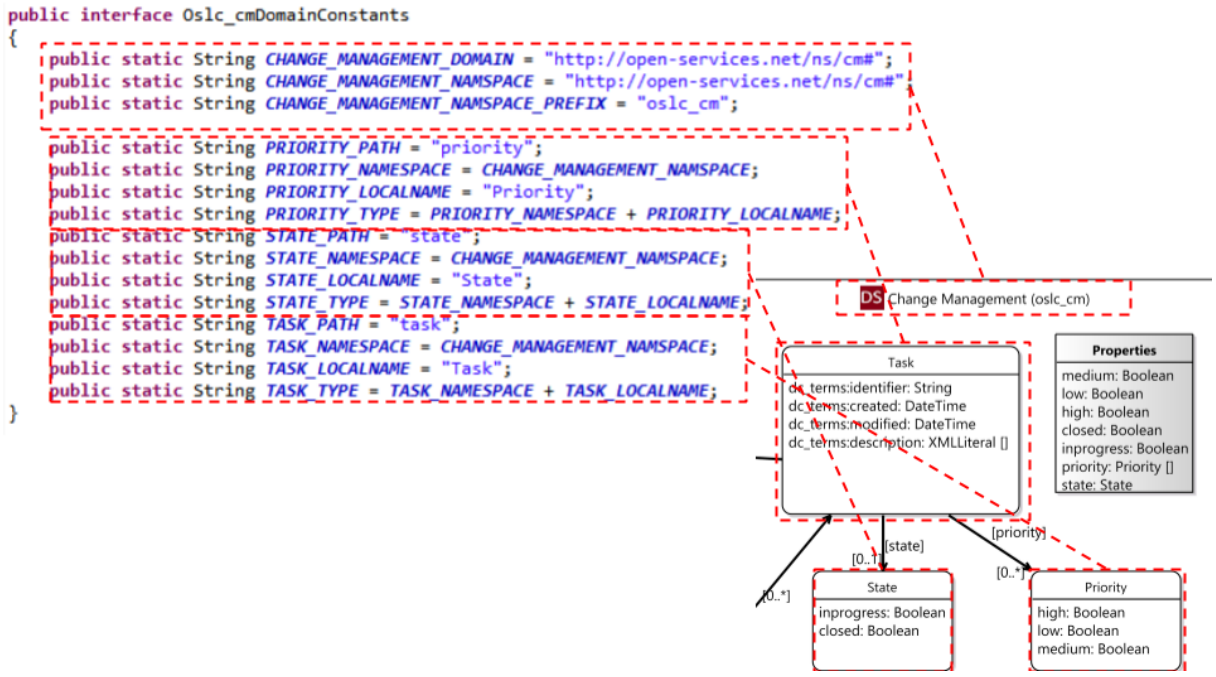


Figura 24 – Relação entre Domínios OSLC e Propriedades Geradas no Lyo

Fonte: O autor.

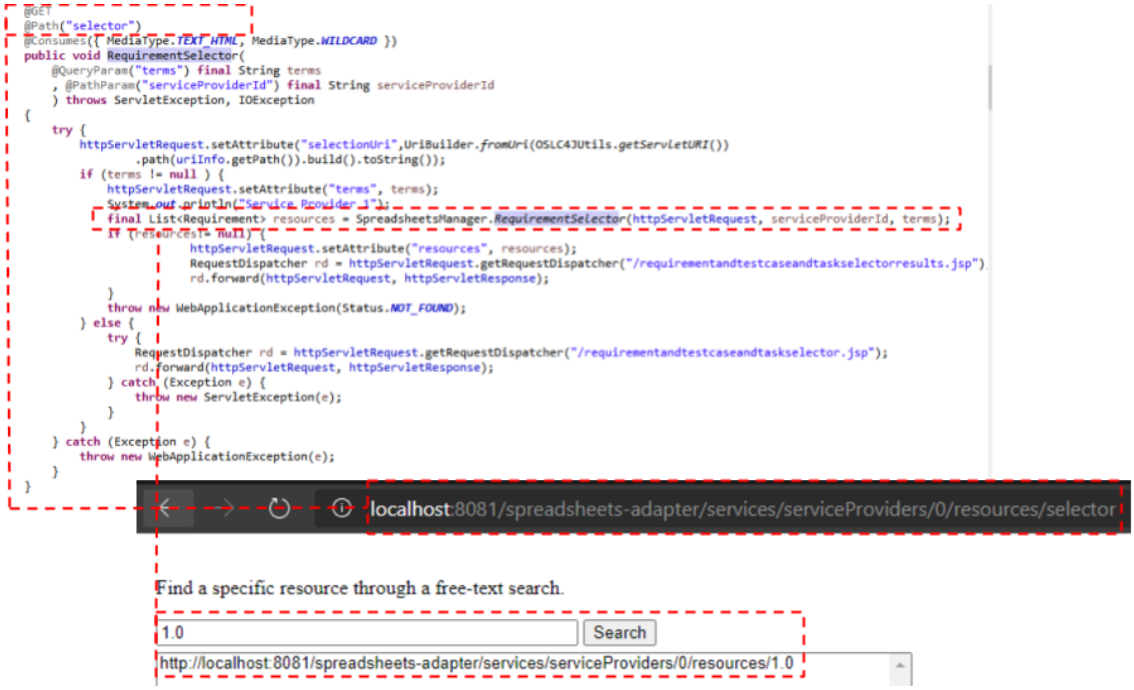


Figura 25 – Relação Entre Busca de Artefatos e Código-fonte

Fonte: O autor.

parâmetro para buscar os requisitos e popular a lista de *resources*. A execução dessa atividade gera uma requisição *Get* com os parâmetros passados pelo usuário que retornam as URIs dos requisitos. Com a customização dessas interfaces, é possível visualizar esses requisitos de forma detalhada acessado a URI correspondente.

5.5 Lições do Capítulo

Este Capítulo fornece os detalhes do uso e do funcionamento da ferramenta Eclipse Lyo. Além disso, descreve o cenário de integração, com base no ecossistema de software da *dtic*.

A partir deste estudo foi possível estabelecer um processo para projetar e representar graficamente os adaptadores no Lyo. Para isso, primeiro são descritas as atividades que foram realizadas para projetar o adaptador, com base nas especificações OSLC e características das ferramentas. Posteriormente, essas decisões de projeto foram modeladas no Eclipse Lyo em suas três perspectivas, que possibilitaram a geração do código-fonte.

6 CONSIDERAÇÕES FINAIS

Este trabalho analisa quatro questões de pesquisa sobre o estado da arte e da prática na integração de ferramentas, buscando as fases de ALM, as ferramentas mais utilizadas, processos, contextos de desenvolvimento, faceta de contribuição e o método de avaliação, bem como vantagens e desvantagens do OSLC.

Identificou-se que o OSLC é motivado por integrações ponto a ponto e, para isso, é necessário criar interfaces que possibilitem a troca de dados entre aplicações chamados de adaptadores. Os adaptadores visam transformar os dados de uma aplicação para o formato de outra. Entretanto, a criação desses adaptadores não é uma tarefa trivial, sendo custosa para as organizações. É necessário filtrar as ferramentas, os artefatos que serão compartilhados, os atributos que serão representados pelos domínios OSLC e os relacionamentos entre as ferramentas. Para isso, as abordagens MDE possibilitam abstrair a complexidade desses ambientes.

Ainda, a literatura mostra que é do interesse da indústria utilizar abordagens MDD e MDE para gerar interfaces conectoras. Ao longos dos anos essas abordagens vem se consolidando como a melhor alternativa para integrar ferramentas com OSLC. Apesar disso, a geração de código-fonte não é 100% automática e ainda não atingiu o nível de cobertura de integrar todo o ambiente. Este cenário pode estar relacionado com o desafio técnico necessário para integrar tais aplicações.

Além disso, esse trabalho apresentou um estudo exploratório sobre a ferramenta Eclipse Lyo com o objetivo de implementar adaptadores OSLC. Para isso, foram utilizadas abordagens para geração de código-fonte com base em modelos. Essas abordagens propõem mitigar alguns desafios que envolvem o processo de integração de ferramentas pois permitem desenvolvedores a trabalhar com um nível maior de abstração. Apesar disso, foram encontradas poucas alternativas para o desenvolvimento de adaptadores OSLC com base em DSLs na literatura.

O OSLC propõe a representação global de artefatos mantidos por quaisquer ferramentas de software ao longo dos domínios do projeto. Entretanto, nem todos os atributos dos artefatos são cobertos pelos domínios OSLC. Por exemplo é possível manter o relacionamento entre um requisito e um caso de teste por meio da propriedade *validateBy* no domínio de Gerenciamento de Requisitos mas não é possível acessar quais tarefas estão relacionadas a um determinado caso de teste, pois não existe uma propriedade dedicada a este relacionamento no domínio de Gerenciamento de Qualidade ou Gerenciamento de Configuração e Mudanças.

Dessa forma, o desenvolvimento de adaptadores OSLC por meio do Eclipse Lyo não foi totalmente explorado na área de Engenharia de Software. Além disso, cada cenário de integração em um ciclo de vida de aplicação de uma empresa possui particularidades que devem ser levadas em consideração na hora de tomar as decisões de projeto. Por mais que o estudo tenha considerado representações de ferramentas bastante utilizadas

no desenvolvimento de software, o cenário representado jamais pode refletir um ambiente real de uma empresa.

Observou-se que o aparato ferramental de suporte propicia a geração automática de código-fonte. Entretanto, apenas os esqueletos dos códigos são gerados dessa forma. É necessário implementar, manualmente, o conteúdo dos métodos para que seja estabelecida a comunicação entre as ferramentas provedoras e consumidoras, possibilitando a troca dos artefatos mantidos por elas. Isso caracteriza uma limitação do aparato ferramental investigado, uma vez que a geração de 100% do código poderia ser obtida para OSLC com base na DSL investigada. Apesar disso, a partir desse estudo foi possível identificar os componentes mínimos necessários para implementar um adaptador de ferramentas, e portanto viável para a execução de um estudo exploratório em domínios de Engenharia de Software.

Por fim, o OSLC possui um grande potencial como tecnologia para automatizar todo o ambiente de desenvolvimento devido a suas características como flexibilidade e reuso. Além disso, há o interesse da indústria no desenvolvimento de processos, metodologias e ferramentas motivadas principalmente para reduzir o custo operacional dessas integrações. Apesar disso, foi possível cumprir os objetivos deste trabalho, identificando o estado da arte e da prática em OSLC, e aprender as técnicas e suporte ferramental utilizado na indústria para desenvolver soluções de integração.

6.1 Limitações e Trabalhos Futuros

Além de estudar os conceitos e suporte ferramental que descrevem os padrões representacionais, é importante realizar uma avaliação empírica da viabilidade do uso do OSLC em ambientes organizacionais com a execução de um estudo de caso. Apesar de ser um tipo de pesquisa flexível, é importante seguir um protocolo bem definido. O processo para a condução de estudos de caso é composto por cinco etapas principais: planejamento, preparação para a coleta de dados, coleta de dados, análise dos dados coletados e reporte do estudo de caso.

Com base nos desafios que a área de automação de processos oferece, os trabalhos futuros incluem:

- Aplicar um estudo de caso em uma equipe de desenvolvimento de software do DTIC, aprimorando tecnologias produzidas ao longo do TCC e transferindo o conhecimento derivado para contextos desta organização;

- Explorar DSLs identificadas no SMS para a representação de elementos de integração de ferramentas de ES por meio de OSLC. Além do Eclipse Lyo, existem outras alternativas para implementar soluções de integração de ferramentas que se pretende explorar:

  - **Guaraná -** O Guaraná (FRANTZ; CORCHUELO; ROOS-FRANTZ, 2016) é uma abordagem para o suporte de implementações de soluções *Enterprise Application Integration*. Além disso, Guaraná possui uma linguagem gráfica denominada Guaraná DSL, a qual permite que engenheiros de software projetem soluções de integração independente de plataformas de integração.
  - ***Tool Integration Language (TIL) -*** É um linguagem de domínio específico para cadeia de ferramentas (BIEHL, 2011). O TIL serve para diversos propósitos, entre eles a geração de códigos de adaptadores a partir da representação de modelos independente da tecnologia, inclusive OSLC.

- Explorar a integração de arquivos por meio de transformações de modelo para modelo e modelo para texto, portanto em contexto de *toolchain* para contextos de DSL;

- Aprofundar os conceitos de representações comuns de ativos de software para diferentes áreas do conhecimento como:

  - *Reusable Asset Specification (RAS)* (BASSO et al., 2013), o core representacional de componentes de software;
  - *Asset Management Specification (AMS)* (BASSO et al., 2016), o core representacional da OSLC para descrição de dados de ferramentas de ES;
  - *Reference Architectural Model Industrie 4.0 (RAMI 4.0)* (WEYRICH; EBERT, 2016), o core representacional de ferramentas no contexto de automação industrial e internet das coisas;
  - RAS++ (BASSO, 2017), o core representacional de ativos de MDD.

- Aprofundar os conceitos de Engenharia de Software Contínua (CSE), estabelecendo um novo estudo que busca identificar as vantagens e desvantagens entre as várias práticas de automação de processos;

- Publicar artigos derivados do TCC.

6.2 Produção Científica

A tabela 13 apresenta a produção científica obtida ao longo do processo de desenvolvimento deste trabalho de conclusão. São cinco materiais escritos como primeiro autor, e quatro como colaborador. Os materiais submetidos encontram-se em processo de revisão por pares nas respectivas conferências. Já o primeiro artigo, este encontra-se em estágio final de refinamento de inglês e será submetido para a revista internacional *Information and Software Technology (IST)*.

Contribuição	Tipo	Título	Local	Estágio
Primeiro Autor	Revista	Open Services for Lifecycle Collaboration: A Systematic Mapping Study	IST	Proof reading
Primeiro Autor	Conferência	Exploring Tool Integration for Continuous Software Engineering with the OSLC and Eclipse Lyo	SBQS 2020	Peer-review
Segundo Autor	Conferência	Mapping Recommendation Approaches for Opportunistic Design Contexts	SBCARS 2020	Peer-review
Terceiro Autor	Conferência	Towards Opportunistic Design of Model Transformation Chains Using Recommendation Services Built on RAS++	SBCARS 2020	Peer-review
Terceiro Autor	Conferência	Validating a Model for Data Extraction Applied to Knowledge Intensive Resources Derived from SMS	SBCARS 2020	Peer-review
Primeiro Autor	Escola Regional	OSLC: Um Estudo de Mapeamento Sistemático	ERES 2020	Peer-review
Primeiro Autor	Escola Regional	Investigando a Integração de Ferramentas com OSLC Através do Eclipse Lyo	ERES 2020	Peer-review
Segundo Autor	Escola Regional	Investigando Técnicas de Recomendação de Assets Híbridos de Software: Um Mapeamento Sistemático	ERES 2020	Peer-review
Primeiro Autor	Resumo Expandido	Open Services for Lifecycle Collaboration: Um Estudo de Mapeamento Sistemático	SIEPE 2019	Apresentado

Tabela 13 – Produção Científica de Materiais de Divulgação de Resultados

REFERÊNCIAS

ADJEPON-YAMOAH, D.; ROMANOVSKY, A.; ILIASOV, A. A reactive architecture for cloud-based system engineering. In: **Proceedings of the 2015 International Conference on Software and System Process**. New York, NY, USA: ACM, 2015. (ICSSP 2015), p. 77–81. ISBN 978-1-4503-3346-7. Disponível em: <http://doi.acm.org/10.1145/2785592.2785611>. Citado 2 vezes nas páginas 47 e 49.

AICHERNIG, B. K. et al. Integration of requirements engineering and test-case generation via oslc. In: **2014 14th International Conference on Quality Software**. [S.l.: s.n.], 2014. p. 117–126. ISSN 1550-6002. Citado 2 vezes nas páginas 36 e 47.

AIELLO, B. **Agile Application Lifecycle Management: Using DevOps to Drive Process Improvement**. Addison-Wesley Professional, 2016. ISBN 0321774108. Disponível em: <https://www.xarg.org/ref/a/0321774108/>. Citado na página 33.

ALVAREZ-RODRíGUEZ, J. et al. Enabling system artefact exchange and selection through a linked data layer. In: **Journal of Universal Computer Science**. [s.n.], 2018. v. 24, n. 11, p. 1536–1560. Cited By 1. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063047015&doi=10.3217%2fjucs-024-11-1536&partnerID=40&md5=28e84d34c63fe49ca6c4021c2f5dbc2b>. Citado 2 vezes nas páginas 46 e 54.

AMS. **Asset Management Specification**. 2014. Disponível em: <http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/>. Acesso em: 08/09/2020. Citado na página 34.

AOYAMA, M. et al. Promis: A management platform for software supply networks based on the linked data and oslc. In: **2013 IEEE 37th Annual Computer Software and Applications Conference**. [S.l.: s.n.], 2013. p. 214–219. ISSN 0730-3157. Citado 2 vezes nas páginas 47 e 49.

AOYAMA, M. et al. A resource-oriented services platform for managing software supply chains and its experience. In: **2014 IEEE International Conference on Web Services**. [S.l.: s.n.], 2014. p. 598–605. Citado 2 vezes nas páginas 47 e 49.

ARNOULD, V. Using model-driven approach for engineering the system engineering system. In: **2018 13th Annual Conference on System of Systems Engineering (SoSE)**. [S.l.: s.n.], 2018. p. 608–614. Citado 2 vezes nas páginas 46 e 50.

BASSO, F. P. **RAS++: Representing Hybrid Reuse Assets for MDE as a Service. Av at <www.cos.ufrj.br/uploadfile/publicacao/2811.pdf>**. Tese (Doutorado), September 2017. Citado 2 vezes nas páginas 29 e 79.

BASSO, F. P. et al. Analysis of asset specification languages for representation of descriptive data from mde artifacts. In: **International Conference on {ENTERprise} Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / {HCist} 2016, October 5-7**. [S.l.: s.n.], 2016. (CENTERIS'16). Citado na página 79.

BASSO, F. P. et al. A common representation for reuse assistants. In: **13th International Conference on Software Reuse**. [S.l.: s.n.], 2013. (ICSR'13), p. 283–288. Citado na página 79.

BAUMGART, A.; ELLEN, C. A recipe for tool interoperability. In: **2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.: s.n.], 2014. p. 300–308. Citado 2 vezes nas páginas 46 e 54.

BEREZOVSKYI, A.; EL-KHOURY, J.; FERSMAN, E. Linked data architecture for plan execution in distributed cps. In: **2019 IEEE International Conference on Industrial Technology (ICIT)**. [S.l.: s.n.], 2019. p. 1393–1399. Citado na página 46.

BERGER, O. et al. Weaving a semantic web across oss repositories: Unleashing a new potential for academia and practice. In: **International Journal of Open Source Software and Processes**. [s.n.], 2010. v. 2, n. 2, p. 29–40. Cited By 3. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-78651542538&doi=10.4018%2fjossp.2010040103&partnerID=40&md5=7614209201597058054 0be6158ace8ec>. Citado na página 47.

BIEHL, M. **Tool Integration Language (TIL)**. [S.l.], 2011. (Trita-MMK, 2011:14). QC 20111130. Disponível em: <http://www1.md.kth.se/~biehl/files/papers/til.pdf>. Citado na página 79.

BIEHL, M. et al. On the modeling and generation of service-oriented tool chains. **Software & Systems Modeling**, Springer Berlin Heidelberg, v. 13, n. 2, p. 461–480, 2014. ISSN 1619-1366. Citado 3 vezes nas páginas 20, 21 e 39.

BIEHL, M.; EL-KHOURY, J.; TöRNGREN, M. High-level specification and code generation for service-oriented tool adapters. In: **2012 12th International Conference on Computational Science and Its Applications**. [S.l.: s.n.], 2012. p. 35–42. Citado 2 vezes nas páginas 47 e 50.

BIEHL, M.; GU, W.; LOIRET, F. Model-based service discovery and orchestration for oslc services in tool chains. In: BRAMBILLA, M.; TOKUDA, T.; TOLKSDORF, R. (Ed.). **Web Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 283–290. ISBN 978-3-642-31753-8. Citado 2 vezes nas páginas 47 e 50.

BIEHL, M. et al. Efficient construction of presentation integration for web-based and desktop development tools. In: **2013 IEEE 37th Annual Computer Software and Applications Conference Workshops**. [S.l.: s.n.], 2013. p. 697–702. Citado na página 47.

BIRO, M. Open services for software process compliance engineering. In: GEFFERT, V. et al. (Ed.). **SOFSEM 2014: Theory and Practice of Computer Science**. Cham: Springer International Publishing, 2014. p. 1–6. ISBN 978-3-319-04298-5. Citado na página 47.

BIRÓ, M. et al. Towards automated traceability assessment through augmented lifecycle space. In: KREINER, C. et al. (Ed.). **Systems, Software and Services Process Improvement**. Cham: Springer International Publishing, 2016. p. 94–105. ISBN 978-3-319-44817-6. Citado 2 vezes nas páginas 47 e 50.

BIRÓ, M. et al. Graceful integration of process capability improvement, formal modeling and web technology for traceability. In: STOLFA, J. et al. (Ed.). **Systems, Software and Services Process Improvement**. Cham: Springer International Publishing, 2017. p. 381–398. ISBN 978-3-319-64218-5. Citado 2 vezes nas páginas 46 e 50.

BROWN, A. W.; PENEDO, M. H. An annotated bibliography on integration in software engineering environments. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 17, n. 3, p. 47–55, jul. 1992. ISSN 0163-5948. Disponível em: <http://doi.acm.org/10.1145/140938.140944>. Citado na página 33.

BUFFONI, L.; POP, A.; MENGIST, A. Traceability and impact analysis in requirement verification. In: **Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools**. New York, NY, USA: ACM, 2017. (EOOLT '17), p. 95–98. ISBN 978-1-4503-6373-0. Disponível em: <http://doi.acm.org/10.1145/3158191.3158207>. Citado 2 vezes nas páginas 46 e 49.

CHEN, J. et al. An open source lifecycle collaboration approach supporting internet of things system development. In: **2019 14th Annual Conference System of Systems Engineering (SoSE)**. [S.l.: s.n.], 2019. p. 63–68. Citado na página 46.

EBERT, C. Improving engineering efficiency with plm/alm. **Software & Systems Modeling**, v. 12, n. 3, p. 443–449, Jul 2013. Citado 2 vezes nas páginas 19 e 38.

EL-KHOURY, J. Lyo code generator: A model-based code generator for the development of oslc-compliant tool interfaces. In: **SoftwareX**. [s.n.], 2016. v. 5, p. 190 – 194. ISSN 2352-7110. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2352711016300267>. Citado 3 vezes nas páginas 46, 50 e 60.

EL-KHOURY, J.; BEREZOVSKYI, A.; NYBERG, M. An industrial evaluation of data access techniques for the interoperability of engineering software tools. In: **Journal of Industrial Information Integration**. [s.n.], 2019. ISSN 2452-414X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2452414X18300359>. Citado na página 46.

EL-KHOURY, J.; EKELIN, C.; EKHOLM, C. Supporting the linked data approach to maintain coherence across rich emf models. In: WĄSOWSKI, A.; LÖNN, H. (Ed.). **Modelling Foundations and Applications**. Cham: Springer International Publishing, 2016. p. 36–47. ISBN 978-3-319-42061-5. Citado na página 47.

ELAASAR, M.; CONALLEN, J. Design management: A collaborative design solution. In: GORP, P. V.; RITTER, T.; ROSE, L. M. (Ed.). **Modelling Foundations and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 165–178. ISBN 978-3-642-39013-5. Citado na página 47.

ELAASAR, M.; NEAL, A. Integrating modeling tools in the development lifecycle with oslc: A case study. In: MOREIRA, A. et al. (Ed.). **Model-Driven Engineering Languages and Systems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 154–169. ISBN 978-3-642-41533-3. Citado na página 47.

FITZGERALD, B.; STOL, K.-J. Continuous software engineering: A roadmap and agenda. **Journal of Systems and Software**, v. 123, p. 176 – 189, 2017. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121215001430>. Citado na página 20.

FOWLER, M.; PARSONS, R. **Domain-specific languages**. [S.l.]: Addison-Wesley, 2011. Citado 2 vezes nas páginas 21 e 37.

FRANTZ, R. Z.; CORCHUELO, R.; ROOS-FRANTZ, F. On the design of a maintainable software development kit to implement integration solutions. **Journal of Systems and Software**, v. 111, n. 1, p. 89–104, 2016. Citado na página 79.

FUGGETTA, A.; NITTO, E. D. Software process. In: **36th International Conference on Software Engineering**. [S.l.: s.n.], 2014. (ICSE '14), p. 1–12. Citado na página 22.

GALLINA, B.; NYBERG, M. Pioneering the creation of iso 26262-compliant oslc-based safety cases. In: **2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)**. [S.l.: s.n.], 2017. p. 325–330. Citado 2 vezes nas páginas 46 e 50.

GALLINA, B.; PADIRA, K.; NYBERG, M. Towards an iso 26262-compliant oslc-based tool chain enabling continuous self-assessment. In: **2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)**. [S.l.: s.n.], 2016. p. 199–204. Citado 3 vezes nas páginas 46, 50 e 54.

GARG, P. K.; JAZAYERI, M. **Process-Centered Software Engineering Environments**. Washington, DC, USA: IEEE Computer Society Press, 1995. ISBN 0818671033. Citado na página 20.

GIANDOMENICO, B. D. et al. Leonardo-finmeccanica - aircraft division needs for integrated systems engineering: The crystal user experience. In: **CEUR Workshop Proceedings**. [s.n.], 2016. v. 1728, p. 95–102. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84999274128&partnerID=40&md5=a8be9e6fb6c55ab0a075a85ea9110d59>. Citado 2 vezes nas páginas 47 e 50.

GüRDüR, D. et al. Knowledge representation of cyber-physical systems for monitoring purpose. In: **Procedia CIRP**. [s.n.], 2018. v. 72, p. 468 – 473. ISSN 2212-8271. 51st CIRP Conference on Manufacturing Systems. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2212827118301173>. Citado 2 vezes nas páginas 46 e 54.

HOHPE, G. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. Addison-Wesley Professional, 2003. ISBN 9780321200686. Disponível em: <https://www.xarg.org/ref/a/0321200683/>. Citado 2 vezes nas páginas 20 e 33.

ILIASOV, A. et al. Formalisation-driven development of safety-critical systems. In: **2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)**. [S.l.: s.n.], 2016. p. 165–172. Citado na página 47.

KACIMI, O. et al. Creating a reference technology platform: Performing model-based safety analysis in a heterogeneous development environment. In: **2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.: s.n.], 2014. p. 645–656. Citado na página 47.

KAISER, C.; HERBST, B. Smart engineering for smart factories: How oslc could enable plug & play tool integration. In: **Mensch und Computer 2015 - Workshop**. [s.n.], 2015. p. 269–280. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029411526&partnerID=40&md5=a32578cbd9ca9de35b4b027fa454271b>. Citado 2 vezes nas páginas 47 e 50.

KERN, M. et al. Integrating static code analysis toolchains. In: **2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2019. v. 1, p. 523–528. Citado 2 vezes nas páginas 46 e 50.

KLEPPE, A.; WARMER, J.; BAST, W. **MDA Explained: The Model Driven Architecture: Practice and Promise**. [S.l.]: Addison-Wesley Professional, 2003. Citado 2 vezes nas páginas 19 e 38.

KLESPITZ, J.; BíRó, M.; KOVáCS, L. Augmented lifecycle space for traceability and consistency enhancement. In: **2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)**. [S.l.: s.n.], 2016. p. 003973–003977. Citado na página 46.

KLESPITZ, J.; BíRó, M.; KOVáCS, L. Enhanced traceability and consistency with augmented lifecycle space. In: **2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)**. [S.l.: s.n.], 2016. p. 207–212. Citado na página 46.

LEDNICKI, L. et al. Integrating version control in a standardized service-oriented tool chain. In: **2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2016. v. 1, p. 323–328. ISSN 0730-3157. Citado 2 vezes nas páginas 46 e 49.

LEITNER, A.; HERBST, B.; MATHIJSSEN, R. Lessons learned from tool integration with oslc. In: DREGVAITE, G.; DAMASEVICIUS, R. (Ed.). **Information and Software Technologies**. Cham: Springer International Publishing, 2016. p. 242–254. ISBN 978-3-319-46254-7. Citado 2 vezes nas páginas 47 e 59.

LU, J. et al. A service-oriented tool-chain for model-based systems engineering of aero-engines. In: **IEEE Access**. [S.l.: s.n.], 2018. v. 6, p. 50443–50458. Citado 2 vezes nas páginas 46 e 50.

MARKO, N. et al. Combining xtext and oslc for integrated model-based requirements engineering. In: **2015 41st Euromicro Conference on Software Engineering and Advanced Applications**. [S.l.: s.n.], 2015. p. 143–150. ISSN 1089-6503. Citado na página 46.

MARTINO, B. d. et al. Towards a uniform semantic representation of business processes, uml artefacts and software assets. In: **2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)**. [S.l.: s.n.], 2016. p. 543–548. Citado 2 vezes nas páginas 46 e 50.

MATINNEJAD, R.; RAMSIN, R. An analytical review of process-centered software engineering environments. In: **Engineering of Computer Based Systems (ECBS)**. [S.l.: s.n.], 2012. p. 64–73. Citado na página 20.

MESSERSCHMITT, D. G. **Software Ecosystem: Understanding an Indispensable Technology and Industry (The MIT Press)**. The MIT Press, 2005. ISBN 9780262633314. Disponível em: <https://www.xarg.org/ref/a/0262633310/>. Citado 2 vezes nas páginas 19 e 38.

MUSTAFA, N.; LABICHE, Y. Employing linked data in building a trace links taxonomy. In: **ICSOFT 2017 - Proceedings of the 12th International Conference on Software Technologies**. [s.n.], 2017. p. 186–198. Cited By 2. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029284603&partnerID=40&md5=14f892da98028ebb6797e10f1f106fc3>. Citado 3 vezes nas páginas 46, 50 e 54.

MUSTAFA, N.; LABICHE, Y. Using semantic web to establish traceability links between heterogeneous artifacts. In: CABELLO, E. et al. (Ed.). **Software Technologies**. Cham: Springer International Publishing, 2018. p. 91–113. ISBN 978-3-319-93641-3. Citado na página 46.

NARDONE, R. et al. An oslc-based environment for system-level functional testing of ertms/etcs controllers. In: **Journal of Systems and Software**. [s.n.], 2020. v. 161, p. 110478. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121219302523>. Citado 2 vezes nas páginas 46 e 50.

OSLC. **Open Services for Lifecycle Collaboration Primer Web Page.** 2020. Acessado em Setembro de 2020. Disponível em: <https://open-services.net/>. Citado na página 20.

OSLC. **OSLC Core Specification**. 2020. Disponível em: <https://archive.open-services.net/bin/view/Main/OslcCoreSpecification.html>. Acesso em: 08/09/2020. Citado na página 35.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, v. 64, p. 1 – 18, 2015. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584915000646>. Citado na página 41.

PIKUS, Y. et al. Semi-automatic ontology-driven development documentation: Generating documents from rdf data and dita templates. In: **Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing**. New York, NY, USA: ACM, 2019. (SAC '19), p. 2293–2302. ISBN 978-1-4503-5933-7. Disponível em: <http://doi.acm.org/10.1145/3297280.3297508>. Citado 2 vezes nas páginas 46 e 49.

PRODANOV, E. C. d. F. C. C. **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico - 2a Edição. [S.l.]**. [S.l.]: Editora Feevale, 2013. Citado na página 22.

REGAN, G. et al. A traceability process assessment model for the medical device domain. In: BARAFORT, B. et al. (Ed.). **Systems, Software and Services Process Improvement**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 206–216. ISBN 978-3-662-43896-1. Citado 2 vezes nas páginas 47 e 54.

REGAN, G. et al. Assessing traceability - practical experiences and lessons learned. In: **Journal of Software: Evolution and Process**. [s.n.], 2015. v. 27, n. 8, p. 591–601. Cited By 6. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84939543650&doi=10.1002%2fsmr.1728&partnerID=40&md5=9a5dea442091d0d2d3d54edac74cc53d>. Citado 3 vezes nas páginas 47, 49 e 54.

RENé, E.; MARTIN, E. Oslc based approach for product appearance structuring. In: **Proceedings of the International Conference on Engineering Design, ICED**. [s.n.], 2017. v. 4, n. DS87-4, p. 259–266. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029763443&partnerID=40&md5=7962fc5cda03cbd3906d735534599e77>. Citado na página 46.

RYMAN, A.; HORS, A. L.; SPEICHER, S. Oslc resource shape a language for defining constraints on linked data. In: **CEUR Workshop Proceedings**. [s.n.], 2013. v. 996. Cited By 12. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84916627887&partnerID=40&md5=714ac7f0d20d83c7e56598ec8356022c>. Citado na página 47.

SAADATMAND, M.; BUCAIONI, A. Oslc tool integration and systems engineering – the relationship between the two worlds. In: **2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications**. [S.l.: s.n.], 2014. p. 93–101. Citado na página 47.

SCHWABER, C. et al. The changing face of application life-cycle management. **Forrester Research**, v. 18, 2006. Citado 3 vezes nas páginas 19, 34 e 38.

SHAHROKNI, A.; SöDERBERG, J. Beyond information silos challenges in integrating industrial model-based data. In: **CEUR Workshop Proceedings**. [s.n.], 2015. v. 1406, p. 63–72. Cited By 2. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84938604236&partnerID=40&md5=d8ba0f35c028cb968d02e7e6f15889c8>. Citado na página 47.

SHARIATZADEH, N. et al. Using linked data with information standards for interoperability in production engineering. In: **Procedia CIRP**. [s.n.], 2016. v. 41, p. 502 – 507. ISSN 2212-8271. Research and Innovation in Manufacturing: Key Enabling Technologies for the Factories of the Future - Proceedings of the 48th CIRP Conference on Manufacturing Systems. Disponível em: <http://www.sciencedirect.com/science/article/pii/S221282711600038X>. Citado na página 46.

SHARIATZADEH, N. et al. Integration of digital factory with smart factory based on internet of things. In: **Procedia CIRP**. [S.l.: s.n.], 2016. v. 50, p. 512–517. Citado na página 47.

SIEBENMARCK, O. Visualizing cross-tool alm projects as graphs with the open service for lifecycle collaboration. In: HASSELBRING, W.; EHMKE, N. C. (Ed.). **Software Engineering 2014**. Bonn: Gesellschaft für Informatik, 2014. p. 125–129. Citado na página 47.

THOMAS, I.; NEJMEH, B. A. Definitions of tool integration for environments. **IEEE Softw.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 9, n. 2, p. 29–35, mar. 1992. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/52.120599>. Citado 2 vezes nas páginas 19 e 33.

TROUBITSYNA, E. Multi-concern integrated engineering of dependable intelligent systems. In: **2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology**

Congress (DASC/PiCom/CBDCom/CyberSciTech). [S.l.: s.n.], 2019. p. 710–715. Citado na página 46.

TüZüN, E. et al. Adopting integrated application lifecycle management within a large-scale software company: An action research approach. **Journal of Systems and Software**, v. 149, p. 63 – 82, 2019. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121218302565>. Citado na página 33.

VAGLIANO, I. et al. Tool integration in the aerospace domain: A case study. In: **2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.: s.n.], 2017. v. 1, p. 731–736. Citado na página 46.

VANZANDT, L. Enabling rational decision making with provenance-annotated oslc relationships. In: **2015 IEEE International Symposium on Systems Engineering (ISSE)**. [S.l.: s.n.], 2015. p. 346–352. Citado 2 vezes nas páginas 46 e 49.

WASSERMAN, A. I. Tool integration in software engineering environments. In: LONG, F. (Ed.). **Software Engineering Environments**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990. p. 137–149. ISBN 978-3-540-46886-8. Citado 2 vezes nas páginas 20 e 33.

WEYRICH, M.; EBERT, C. Reference architectures for the internet of things. **IEEE Software**, IEEE Computer Society, Los Alamitos, CA, USA, v. 33, n. 01, p. 112–116, jan 2016. ISSN 1937-4194. Citado na página 79.

WICKS, M. N.; DEWAR, R. G. Controversy corner: A new research agenda for tool integration. **J. Syst. Softw.**, Elsevier Science Inc., v. 80, n. 9, p. 1569–1585, set. 2007. ISSN 0164-1212. Citado 2 vezes nas páginas 19 e 38.

WIERINGA, R. et al. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. **Requir. Eng.**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 11, n. 1, p. 102–107, dez. 2005. ISSN 0947-3602. Disponível em: <http://dx.doi.org/10.1007/s00766-005-0021-6>. Citado na página 52.

WOLVERS, R.; SECELEANU, T. Embedded systems design flows: Integrating requirements authoring and design tools. In: **2013 39th Euromicro Conference on Software Engineering and Advanced Applications**. [S.l.: s.n.], 2013. p. 244–251. Citado na página 47.

ZADEH, N. et al. Service oriented integration of distributed heterogeneous it systems in production engineering using information standards and linked data. In: **Modelling and Simulation in Engineering**. [s.n.], 2017. v. 2017. Cited By 1. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85017128670&doi=10.1155%2f2017%2f9814179&partnerID=40&md5=19b235baf1fb6d45687656fb389fa9e3>. Citado na página 46.

ZHANG, W. et al. Towards tool integration through artifacts and roles. In: **2012 19th Asia-Pacific Software Engineering Conference**. [S.l.: s.n.], 2012. v. 1, p. 603–613. Citado na página 47.

ZHANG, W.; MOLLER-PEDERSEN, B. Modeling of tool integration resources with oslc support. In: **Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on**. [S.l.: s.n.], 2014. p. 99–110. Citado na página 20.

ZHANG, W.; MøLLER-PEDERSEN, B. Establishing tool chains above the service cloud with integration models. In: **2013 IEEE 20th International Conference on Web Services**. [S.l.: s.n.], 2013. p. 372–379. Citado 2 vezes nas páginas 47 e 50.

ZHANG, W.; MøLLER-PEDERSEN, B. **Modeling of tool integration resources with OSLC support**. 2014. 99-110 p. Citado 4 vezes nas páginas 21, 39, 46 e 54.

ZHANG, W.; MøLLER-PEDERSEN, B.; BIEHL, M. A light-weight tool integration approach: From a tool integration model to oslc integration services. In: **ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends**. [s.n.], 2012. p. 137–146. Cited By 8. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84868691539&partnerID=40&md5=ff30ae7cc6914be880e2eb77a11abe8a>. Citado na página 47.

Apêndices

APÊNDICE A – ESTUDO DE MAPEAMENTO SISTEMÁTICO

Open Services for Lifecycle Collaboration: A Systematic Mapping Study

Anonymous Author(s)

ABSTRACT

Background: Open Services for Lifecycle Collaboration (OSLC) is an open standard for tool interoperability, which allows data federation throughout Software Engineering (SE) application lifecycles. The OSLC community has been active since 2008 there still an open question: "What is the state of the art and practice of OSLC for tool integration in Application Lifecycle Management (ALM) for Software Engineering environments?". *Aims*: In order to characterize the state of the art and practice in the area, our goal is to map all approaches adopting OSLC in SE lifecycles. *Method*: This paper presents a Systematic Mapping Study (SMS) analyzing 59 primaries studies and addressing integration issues, in special to SE toolchains. *Results*: Our findings show that OSLC has been mostly applied to the safety-critical industry and is often employed in Requirements, Analysis & Design, Change & Configuration Management and Test domains. *Conclusions*: The main OSLC advantages are related to linked data, involving not only tool adapters for point-to-point integrations, but also proposing approaches for tool replacement in the toolchain, as well as including modification of OSLC domain specifications and solutions for automated activities for tool integration.

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; **Development frameworks and environments**; **Software as a service orchestration system**.

KEYWORDS

Open Services for Lifecycle Collaboration, OSLC, Application Lifecycle Management, Tool Integration, Systematic Mapping Study

ACM Reference Format:

Anonymous Author(s). 2018. Open Services for Lifecycle Collaboration: A Systematic Mapping Study. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Software tools are used to support teams throughout the software development lifecycle. Yet, fully automated integrated environments are rare in the industry due to the number of features involved in the tool interoperability process [82]. Tools are usually designed without the needed support for integration with other software tools [86]. Besides, such tools have different configurations, functionalities, vendors, and handle diverse kinds of artifacts. In addition, tool integration scenarios add challenges to achieving an end-to-end integrated environment, such as traceability among tool artifacts, thus characterizing a complex interoperation issue.

In Software Engineering (SE), the main challenge of tool integration is to define how tools, embedded with different features, can work together towards a common goal. To achieve a common goal, tools might need to use the same type of data, user interface conventions, and functionalities [76]. Regarding data integration, there is a range of different approaches, options, and challenges to create, manage, store, reuse, and access data across applications [81].

An alternative to integrate data into heterogeneous tools is to manually transform, synchronize, and update SE artifacts. Hence, the Application Lifecycle Management (ALM) paradigm takes a different approach, allowing an automated flow of data sharing generated during the software development lifecycle. ALM is a paradigm that integrates and manages governance, development, and maintenance activities gathering developers, stakeholders, and end-users in a complex environment that promotes collaboration throughout project domains. ALM also includes the chaining of several different tools such as word processors, compilers, version control management systems, model designers, and task control systems, all of them consuming and producing information stored as software artifacts.

ALM is essential in modern approaches for Continuous Software Engineering (CSE) integration. These approaches are generally implemented through the point-to-point topology [30], in a direct connection between two tools [38]. In other words, tools used in the Software Engineering lifecycle are typically tied to specific interfaces. In the past, this coupling of tool was a need for operationalizing the process automation through Process-centered Software Engineering Environments (PSEEs) [35]. However, since such integrations were characterized by tightly coupled tools, when a tool is discontinued and/or is simply replaced in the process, this implies rework and loss of traces in the flow of information throughout the ALM. Thus, once the high-coupling is one of the reasons why PSEEs are unfeasible for adoption by the software industry [53], alternatives for low tool coupling are necessary.

In answer to this limitation, taking into account ALM integration scenarios, some works, such as [13], propose the low coupling through provisioning of assets as services. The main goal is to minimize problems related to the tool integration process adopted in Software Engineering contexts through a typical architecture for services [88]. Hence, to address the shortcomings of exists solutions, the Open Services for Lifecycle Collaboration has been proposed [60]: an industrial specification, as an open standard for tool interoperability, allowing data federation throughout the execution of

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

a software project. OSLC defines a standard representation model to artifacts, as well as methods that allow data exchange between tools.

To our knowledge, there is no study mapping the state-of-the-art and their practical approaches of OSLC, thus two gaps that hampers decision making. As consequence, software engineers face decision making difficulties when selecting approaches to the OSLC integration practice. These difficulties include the lack of summarized data for analysis of ALM scenarios scoping integration requirements, as well as data highlighting benefits and drawbacks of OSLC tool integration. To the researcher point of view, it is also difficult to find out existing contributions in the area to decide whether they should develop a new contribution/increments in the state-of-the-art. A third research gap is the lack of a replicable study, so that software engineers could replicate studies along years to follow the evolution of the research area. In this paper we sought to bridge these three gaps found in the literature. This article contributes with a Systematic Mapping Study (SMS) [62] of studies devoted to support the application of OSLC standard, analyzing the integration issues associated with SE toolchains in ALM contexts, and deriving the maturity and research trends in the area.

The remainder of the paper is organized as follows. In Section 2, we present background concepts. Section 3 describes the SMS protocol. Section 4 show the SMS results and discuss our findings, and Section 6 the threats to validity. Finally, Section 8 concludes the paper.

2 BACKGROUND

The software process automation through tool integration is not a trivial task. In order to well characterize the studies devoted for tool integration with OSLC in SE projects, this section introduces the main concepts founding our literature review. In the following, we present an introduction to ALM, followed by a discussion on the integration type discussed along the paper and a high-level conceptual demonstration for tool integration to provide an appropriate motivation of this study.

2.1 Application Lifecycle Management

As illustrates Figure 1, Application Lifecycle Management (ALM) is a paradigm for integrating and managing activities related to the governance, development, and maintenance of software products [78]. Software development activities are linked to software engineering disciplines such as Business Modeling, Requirements, Implementation and others. Different process models suggest different ALM disciplines. In addition, a complete configured development environment with tools supporting collaboration in software engineering tasks provide the necessary structure for the entire software development effort, including specific details about tasks, roles, responsibilities, and essential milestones helping stakeholders on tracking the progress of the integrated process [3]. To this end, ALM embraces the idea of end-to-end integration of application lifecycles, avoiding information silos due to characteristics such as traceability, visibility, and process automation [69].

To tame the complexity of such environments, several tools assisting Software Engineering tasks have been proposed. For example, one type of tool is devoted to represent the software process,

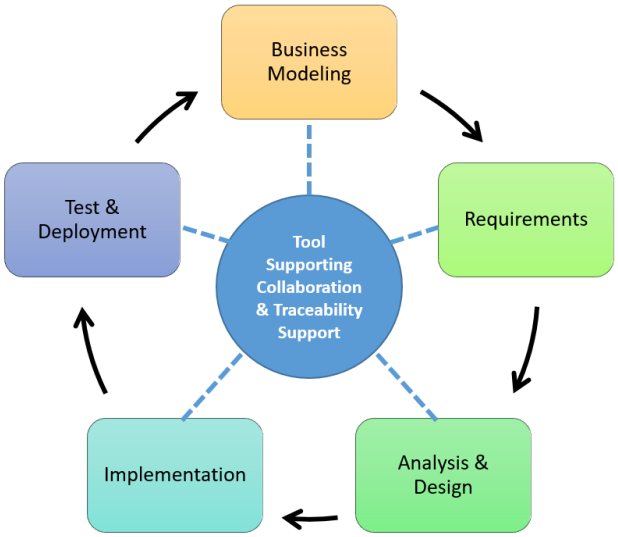


Figure 1: Application Lifecycle Management in the Context of Software Engineering Tool Integration Connecting Five Software Engineering Disciplines

as illustrates Figure 2. The tool adopted to represent the illustrated software process is called EPF Composer, and it is built on Domain Specific Language (DSL) concepts.

Since 2000, as a mean to reduce issues related to integration of software artifacts along many phases of Application Lifecycle Management (ALM), standard representations for models, metamodels and transformations have been proposed in the literature of the area [44]. This scenario demands an integration known as "file integration", and it can become more complex if the tools are heterogeneous. So far, the desired homogeneity of modeling artifacts have failed, incurring in persistent integration issues [49].

As illustrates Figure 2, even if these modeling tools, such as the Astah UML, have succeed in a unique representation for all the artifacts produced along the application lifecycle, software development processes are assisted by many other types of tools than those devoted to modeling, such as the BugZilla, an issue tracking system for bug reporting. Due to different nature of these tools, some ALM instances requires from software engineers some manual configurations of toolchains to connect inputs and outputs, for example, between the tasks "Detail System-Wide Requirements" with the task "Implement Solution". Tipically, integration in ALM happens externally to the software process, by interoperating heterogeneous and hybrid assets, starting data transformation between participant tools along the execution of process tasks.

Through web services architectural styles and/or file transfer styles, these tool integration solutions allow to automate software development processes. A manual integration effort may be need to connect tools A and B, which implies in costs. On the other hand, integration effort could be reduced through automated approaches for tool chain. Hence, in order to reduce the manual effort for integration, some platforms and standards for services emerged

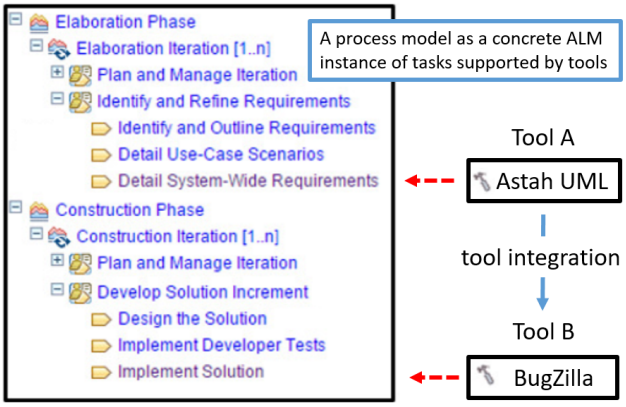


Figure 2: Example of Two OpenUP Tasks Requiring the Integration of Tools for requirements specification and for bug tracking

with the idea to cover the whole lifecycle of a software product through a standard asset representation model, such as the OSLC.

2.2 Point-to-Point Integration Through a Common Asset Specification

In order to characterize the goals of OSLC, this section presents the main concepts associated with this industrial integration standard for software engineering ALM tools, starting by architectural style.

Point-to-point integration topology establishes a direct connection between two tools [38]. Also, this topology is characterized by applications tightly coupled, and so when a tool is replaced, it is necessary to rework on the integration process, resulting in significant operational cost.

Open Services for Lifecycle Collaboration (OSLC) [59] is an industry specification to represent interchangeable ALM assets. As illustrates Figure 3, it is characterized by a point-to-point integration style, where all tools implement the same communication interface defined as the Asset Management Specification (AMS) [5].

Similarly, another specification, such as Reusable Asset Specification (RAS) [39], which is an OMG proposal, allows one to represent details for software artifacts also using the point-to-point architectural style. Accordingly, software component assets (RAS) require different information from tool assets (AMS), such as specialization points that each tool must supply to adapt some artifacts for a specific ALM phase in the software process. Both approaches are capable to represent artifact details such as classification, but only OSLC is able to represent artifact and asset federation in inter-cloud scenarios, i.e., cloud of clouds.

OSLC defines a standard artifact representation model, which are produced throughout the project, as well as methods allowing tools to share data between them. These specifications are defined in the OSLC Core Specification, which consists of rules and standards for creating, updating, retrieving, and linking resource files and the RDF/XML file structure. In addition to the main specification, there are OSLC Domains that extend the Core Specification and define

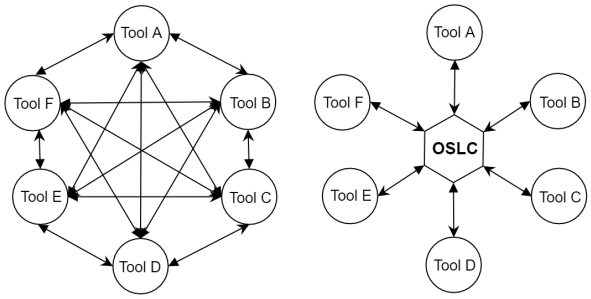


Figure 3: OSLC Toolchain Point-to-point Integration

rules for their respective domains (*e.g.*, Requirements Management, Quality Management, Change Management).

OSLC also provides interoperability throughout different development domains without knowing how the tool operates internally. Within an OSLC toolchain, a tool can be classified as a provider or a consumer, allowing the creation of an inter-connected network of tools in collaboration. Providers are designed to store and provide data, allowing consumers easy access to browse, create, and retrieve data [2].

OSLC is based on linked data principles and follows the rules defined by Tim Berners-Lee [50] to linking data on the Web. In this sense, the relationships among artifacts in a toolchain can be established without duplicating data through links. These links are maintained in a structure named Triple that consists of a subject-predicate-object data that describes the relationship among artifacts, stakeholders, and tasks.

2.3 Example of Tool Integration for one Small ALM Scenario

In this Section we exemplified a scenario for modeling of tool integration in the OSLC context. One can mention several Model-Driven Engineering (MDE) tools designed with the intent of standardizing software artifacts representations [49]. These tools help developers to represent integration details in higher-level than those required in hand-on when developing integration services. Besides, model-based tool support helps in capturing, organizing, and storing most of the information exchanged in a standard formats adopted by Software Engineering contexts.

The exemplified diagrams in this section are associated with an MDE approach, which enable to improve the abstraction level of software integration through Domain Specific Languages (DSLs) [12]. MDE allows the code generation of integration solutions, which can help the assimilation of integration theory to practice through mappings from models to code [31]. The use of DSLs increases productivity and improves communication between domain specialists, as it is more visible what is happening in the system.

In this literature review we found many alternatives for designing tool chains and supporting automatic generation of tool integration resources through MDE. In order to impartially illustrate an ALM integration scenario, we opted for a simple UML diagramming, without using semantics for tool integration from any surveyed.

In this sense, assume an ALM scenario where Software Engineers must connect tools devoted for Requirement Engineering with tools devoted for Testing disciplines, associating roles working for a software development company as responsible for the execution some tasks such as represent test plan and run test cases. In order to illustrate how integration occurs in terms of modeling, we also adopted an example available in the OSLC project, designed with the intent at helping the community interested in integration to adopt this standard specification, and also to develop new compatible solutions.

In order to represent different integration needs, there are three perspectives for the representation of models are usually available in a tool chain design scoping OSLC: Domain Specification View, Toolchain View, and a Workflow/Adapter Interface View. Each perspective allows representing a type of diagram, here illustrated with the UML, as follows:

- Domain Specification View - It is about the representation of the OSLC domains. It is possible to add multiple domains, as shown in Figure 4. Each domain is composed by tools and their domain entities. For instance, assume that the quality management Software Engineering discipline is composed of unique domain represented by a subsystem UML notation named "Testing Tool". This domain owns three resources managed by the testing tool: Test plan, Test case, and Test script. These resources contain properties such as title and description, whose data type definition and value are represented conforming to the RDF format and in conformity with the standard OSLC specification. Other domains are composed by the subsystem of the company owning all the users as roles and a requirement engineering discipline executed in this company through the use of a unique tool. As one can notice, the domain specification view allows to connect essential information offered by these tools through services, e.g., by associating entities test plan and test case from a Testing Tool domain with the entity person from the company repository domain, and the entity test script with the requirement entity from the requirement engineering domain. Thus, domains are Software Engineering disciplines that can group several tools and their compounds analogically to the UML subsystem notation.
- Toolchain View - It refers to the structure of the toolchain and artifacts that will be shared between applications. Figure 5 illustrates a component UML diagram with a scenario integrating all the resources shared by these tools through services and interfaces. Consumer interfaces are represented by the letter and providers follows the same semantic available in the UML, meaning that Test Script consumes services through the interface Requirement. Test Plan and Test Case consume services through the interface Person. In this example, the Testing Tool consumes requirements at the same time that it provides two interfaces to the external components to interact through triggers, for example. In contrast, the Requirement Engineering Tool provides an interface for external tools assessing requirements, i.e., used by the Testing Tool.

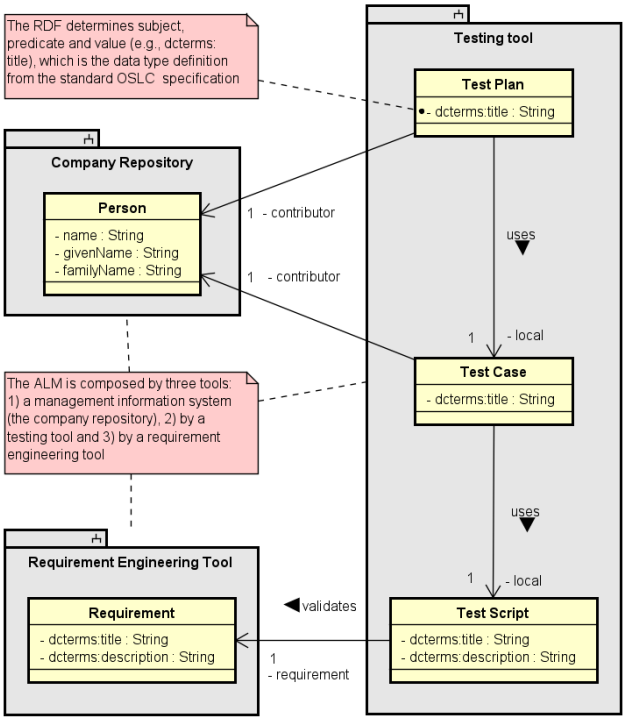


Figure 4: Domain Specification View

- Adapter Interface View - It represents the structure of each tool's adapters represented in the Toolchain perspective to interoperate the resources exchanged along ALM lifecycles. This structure follows the OSLC Core Specification, being the basis for code generation. In this sense, the adapter is the logic behind the overall integration, connecting all the interfaces through a workflow of tasks requesting service calls, as illustrates Figure 6. We adopted the Statechart UML Diagram to illustrate hole of an adapter. It is important to notice that in this example, all the service calls are asynchronously invoked by the adapter (or adapter) conforms triggers are executed the participants of the execution process of their tasks.

2.4 Implementation of Integration Approaches and Design Practices

The toolchain view allows to visualize the collaboration of tools in a level of components and their interfaces. As illustrates Figure 5, in an environment consisting of requirements management tool (Tool A) and a testing management tool (Tool B), a requirement in Tool A could be validated by a test case in Tool B. Besides, considering the implementation of an integration built on OSLC, specific tools can update these implementation links through Restful functions, interchanging data through XML or JSON formats through adapters.

To this level of implementation, there are three approaches providing alternative OSLC support in software tools, including: native

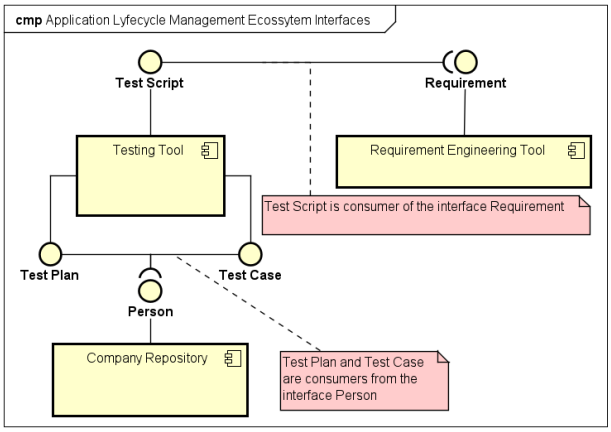


Figure 5: Toolchain View

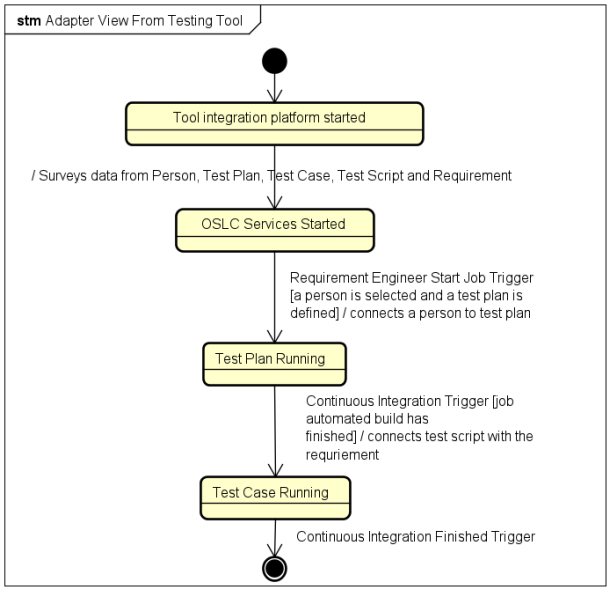


Figure 6: Adapter View

support, plugin and adapter. The native support approach is recommended for tool developers, so that they can prepare their tools for further integration. The plugin and adapter approaches are similar, although the plugin alternative is recommended only in cases where there is knowledge of the technologies involving the construction of the tool, such as programming language and architecture. So, the plugin approach adopts a native integration. In a non native integration scenario, the recommended approach is to build OSLC adapters, the less coupled approach for tool integration.

Considering the design of an integration solution in a sort of automated process, representations of adapters must be drawn up following design decisions, so that they can be activated through triggers (asynchronously) and also synchronously through direct execution. In the example, each transition from a state to another can be implemented as a tool adapter. The overall scenario can be implemented through a workflow automation tool.

Finally, after performing these three modeling views, MDE based approaches allows to generate code for OSLC adapters, thus promoting the automated integration from an independent model to a platform specific representation.

2.5 Motivation

Similarly to the UML example, several other approaches and tools have been proposed in the literature of the area to fulfill needs for reducing costs in integration with the OSLC standard. However, the literature of the area lacks a characterization of these studies discussing their relevance for lifecycle phases, application context and contribution facets devoted mainly to software process toolchain. In the following we present a mapping study analyzing 59 primaries studies reporting OSLC usage.

3 RESEARCH PROTOCOL

In this section, we describe the process applied in our systematic mapping study, illustrated in Figure 7. The protocol followed has been proposed by Petersen *et al.* [62], which defines a process for systematic mapping studies in the Software Engineering area. The study package is available at Google Drive [1]

3.1 Research Questions

The research questions (RQ) were structured to find out how OSLC is used throughout the software lifecycle and are structured as follows:

- **RQ1:** OSLC standard is used in which phases of the software lifecycle? Application integration studies aimed at Software Engineering contexts are typically associated with certain application life cycles. Thus, this research question seeks to characterize the studies contributing to certain SE disciplines, providing a coverage map that allows us to infer whether software development processes are fully automated.
- **RQ2:** What are the contexts of organizations using the OSLC standard? In order to provide a map of the state of practice on the topic, it is also in the interest of this study to understand what kind of organizations have been adopting the OSLC standard. With this in mind, it is expected to establish a relationship between theory and practice in the research area.
- **RQ3:** What are the most frequent types of research about the OSLC standard? In order to understand the maturity of the studies from an empirical point of view, we also seek to classify the studies according to the type of assessment adopted.
- **RQ4:** Which are the facets of the study contribution? Finally, our goal is to characterize the area's contributions according to its facet: whether with tools, modeling, processes and methodologies.

[1] https://drive.google.com/drive/folders/14Fus$_y$ehjlRxRXP53$_V$1ATxgzsdfo7Bd?usp = sharing

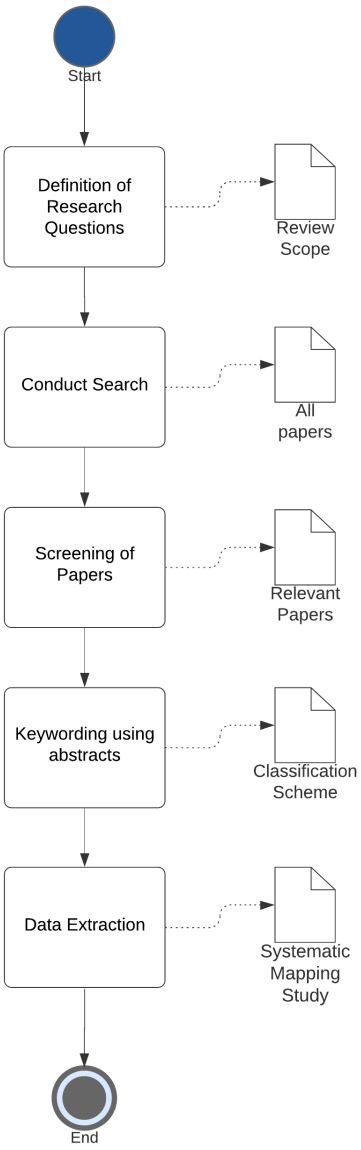


Figure 7: SMS Process

3.2 Search Strategy

The search for papers was carried out in five digital libraries widely used by the scientific community: ACM Digital Library, IEEE Xplore, Springer Link, Science Direct and Scopus.

The search strings are constructed with terms structured to obtain broad coverage of studies about OSLC standard, as shown in Table 1.

3.3 Inclusion and Exclusion Criteria

Inclusion criteria (IC) and exclusion criteria (EC) aim to filter the articles related to the OSLC standard. In our study, articles shall satisfy all ICs to be selected. However, if the article is associated with at least one EC, it is excluded from SMS. The criteria are as follows:

- **IC1:** The study presents in the title, abstract, keywords or introduction the terms "OSLC" or "Open Services for Lifecycle Collaboration".
- **IC2:** The study provides an example use, method, tool, metric, or process that uses the OSLC standard as a solution.
- **EC1:** The study is not in English.
- **EC2:** The study is not primary.
- **EC3:** The study is a short paper (less than four pages).
- **EC4:** The study is not from the Computer Science area.
- **EC5:** The study was published before the year 2008.
- **EC6:** The study is not available for download.
- **EC7:** The study is duplicated.

Table 1: Search Strings

Database	String
ACM DL	("Open Services for Lifecycle Collaboration" OSLC) AND (Integration Interoperability "Linked Data" Lifecycle Traceability)
IEEE Xplore	("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability)
Springer Link	("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability)
Science Direct	("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability)
Scopus	TITLE-ABS-KEY (("Open Services for Lifecycle Collaboration" OR OSLC) AND (Integration OR Interoperability OR "Linked Data" OR Lifecycle OR Traceability))

3.4 Quality Criteria

The Quality Assessment Criteria (CQA) aim to classify articles according to their relevance to the work. The papers present a score for each CQA and after the sum of all scores can be classified as Excellent Quality, Very Good Quality, Good Quality, Bordeline Quality and Low Quality. The Quality Assessment Criteria are as follows:

QC 1: Applicability for Software Engineering (SE) Lifecycles - Applies to estimate the sum of contributions presented by the article, i.e. the weight is cumulative for research question 1:

- Value 3 = Excellent quality - Explored the integration of five or more tools from at least five different SE lifecycle phases.
- Value 2 = Good Quality - Explored the integration of at three tools from three different SE lifecycle phases.
- Value 1 = Borderline Quality - Explore the integration of only three or two tools adopted in two SE lifecycles.
- Value 0 = Low Quality - Does not explore SE lifecycles.

QC 2: Feasibility Report - Applies to question RQ 2 and aims at identifying the relevance of the report considering real needs from organizations.

- Value 2 = Excellent quality - The study presents a complete report of viability in a real organizational scenario.
- Value 1 = Good Quality - The study presents a superficial report of viability in an also real organizational scenario.
- Value 0 = Low Quality - The study does not present a viability report.

QC 3: Evaluation methodology that derived their conclusions - Applies to RQ 3 investigating evaluation type employed by the selected study

- Value 3 = Excellent quality - The study is an evaluation research, i.e., presents results from a controlled experiment, or from a case study, or from an action research.
- Value 2 = Very Good Quality - The study is of type validation research, i.e., presents results from an analytical study, from a survey, from a simulation, or from a focus group.
- Value 1 = Good Quality - The study is a solution proposal, i.e., their conclusion are derived from a proof of concept, also known as conceptual demonstration, of a proposed tool, method or process.
- Value 0.5 = Borderline Quality - The study is an experience report of the own authors on the OSLC context.
- Value 0 = Low Quality - The study is a philosophical paper type.
- Value 0 = Low Quality - The study is of opinion paper type.

QC 4: Contribution facet in MDE tool support - Applies to estimate the contributions providing tool support for representation of integration independent from the OSLC platforms, i.e. the weight is cumulative for research question 4:

- Value 2 = Excellent Quality - Presents own DSL and model transformations to generate OSLC specifications.
- Value 1.5 = Very Good Quality - Presents a third-party DSL and own model transformations to generate OSLC specifications.
- Value 1 = Good Quality - Presents an own DSL.
- Value 0.5 = Borderline Quality - Presents a third-party DSL.
- Value 0 = Low Quality - Does not provide a DSL for integration design.

3.5 Data Extraction

In order to answer the presented research questions and adequately analyze the results, we used the data extraction form shown in Table 2.

4 EXECUTION AND ANALYSIS

This section presents the analysis of the results obtained from our SMS protocol. The results are based on papers selected during the screening process, as pictured in Figure 8. In the first step, the search strings were used in databases were 278 papers returned. Then, the duplicated studies among databases were disregarded, remaining 223 papers. Thus, after applying the IC and EC based on the title, abstract, keywords, and introduction of these papers, 59 studies were considered according to the research topic, listed in Table 3.

Table 2: Data Extraction Form

Item	RQ
Id	-
Title	-
Software lifecycle phases	RQ1
Tools	RQ1
Software development process	RQ1
Advantages	RQ1
Disadvantages	RQ1
Organization Context	RQ2
Research Type	RQ3
Facet	RQ4

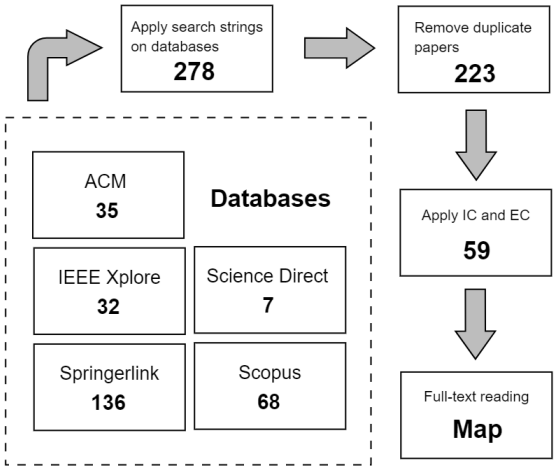


Figure 8: Screening of Papers Process

In our SMS, we have grouped authors and their publications over the years. The first work we found was published in 2010 and reports the research and the potential for a new specification for tool integration, as seen in Table 5. Since then, researchers have explored the OSLC and proposed new approaches to improve tool integration in ALM environments.

4.1 OSLC standard is used in which phases of the software lifecycle? (RQ1)

To answer RQ1, we identified three points of the studies: 1) the SLDC phases in which the OSLC is most frequently used, 2) the software tools associated with the proposed solutions and, 3) software development process adopted by the authors.

The Rational Unified Process (RUP) disciplines are used as categories to map OSLC usage. This decision is due to many software engineers have a previous knowledge about these disciplines. In

Table 3: List of Selected Studies and Application of the Quality Criteria

ID	Paper Reference	Year	CQ1	CQ2	CQ3	CQ4	Score
s1	Baumgart and Ellen [9]	2014	2	2	1	0.5	5.5
s2	Marko et al. [52]	2015	2	2	1	0	5
s3	VanZandt [80]	2015	0	2	0	0	2
s4	Klespitz et al. [46]	2016	0	1	0	0	1
s5	Lednicki et al. [47]	2016	1	2	1	0.5	4.5
s6	Zhang and Møller-Pedersen [90]	2014	2	2	1	2	6
s7	d. Martino et al. [23]	2016	0	1	1	0	2
s8	El-khoury [25]	2016	0	1	0	2	3
s9	Shariatzadeh et al. [72]	2016	2	1	1	0	4
s10	Gürdür et al. [37]	2018	0	2	2	0.5	4.5
s11	Nardone et al. [58]	2020	1	2	1	0.5	4.5
s12	El-khoury et al. [26]	2019	2	2	1.5	0.5	6
s13	Troubitsyna [77]	2019	0	0	0	0	0
s14	Kern et al. [43]	2019	1	2	1	0.5	4.5
s15	Chen et al. [22]	2019	0	1	1	0	2
s16	Berezovskyi et al. [10]	2019	0	1	1	0.5	2.5
s17	Pikus et al. [63]	2019	1	2	1	0	4
s18	Lu et al. [51]	2018	1	2	1	2	6
s19	Arnould [8]	2018	0	1	1	1	3
s20	Mustafa and Labiche [57]	2018	1	2	3	1	7
s21	Alvarez-Rodríguez et al. [4]	2018	1	2	2	0	5
s22	Buffoni et al. [21]	2017	1	1	1	0	3
s23	Gallina and Nyberg [33]	2017	2	1	0	0	3
s24	Gallina et al. [34]	2017	0	1	2	0	3
s25	Vagliano et al. [79]	2017	1	2	1	0.5	4.5
s26	Klespitz et al. [45]	2017	0	0	0	0	0
s27	Mustafa and Labiche [55]	2017	0	0	1	0	1
s28	Biró et al. [19]	2017	2	1	1	0.5	4.5
s29	Zadeh et al. [85]	2017	1	2	3	0.5	6.5
s30	René and Martin [66]	2017	0	1	1	0.5	2.5
s31	Iliasov et al. [40]	2016	0	2	1	0.5	3.5
s32	Leitner et al. [48]	2016	0	2	0.5	0.5	3
s33	Biró et al. [18]	2016	0	1	0	0	1
s34	Shariatzadeh et al. [73]	2016	0	1	1	0	2
s35	El-Khoury et al. [27]	2016	1	2	1	1.5	5.5
s36	Di Giandomenico et al. [24]	2016	3	2	3	0.5	8.5
s37	Adjepon-Yamoah et al. [1]	2015	0	2	1	0.5	3.5
s38	Kaiser and Herbst [42]	2015	0	1	0	0	1
s39	Regan et al. [64]	2015	0	1	2	0	3
s40	Shahrokni and Söderberg [71]	2015	0	2	0.5	0	2.5
s41	Aichernig et al. [2]	2014	1	2	1	0	4
s42	Siebenmarck [74]	2014	0	0	0	0	0
s43	Aoyama et al. [7]	2014	1	2	1	0.5	4.5
s44	Biro [17]	2014	0	0	0.5	0	0.5
s45	Saadatmand and Bucaioni [68]	2014	0	1	0	0.5	1.5
s46	Kacimi et al. [41]	2014	2	2	1	0	5
s47	Wolvers and Seceleanu [84]	2013	2	2	1	0.5	5.5
s48	Elaasar and Neal [29]	2013	1	2	2	1	6
s49	Biehl et al. [16]	2013	0	2	1	1.5	4.5
s50	Elaasar and Conallen [28]	2013	0	0	1	1	2
s51	Aoyama et al. [6]	2013	1	2	0	0.5	3.5
s52	Ryman et al. [67]	2013	0	0	0	0	0
s53	Zhang and Møller-Pedersen [89]	2013	2	2	0	0.5	4.5
s54	Zhang et al. [91]	2012	2	2	1	2	7
s55	Biehl et al. [14]	2012	2	1	1	2	6
s56	Biehl et al. [15]	2012	0	0	1	0.5	1.5
s57	Zhang et al. [87]	2012	2	2	1	2	7
s58	Berger et al. [11]	2010	0	0	0.5	0	0.5
s59	Regan et al. [65]	2014	0	1	1	0	2

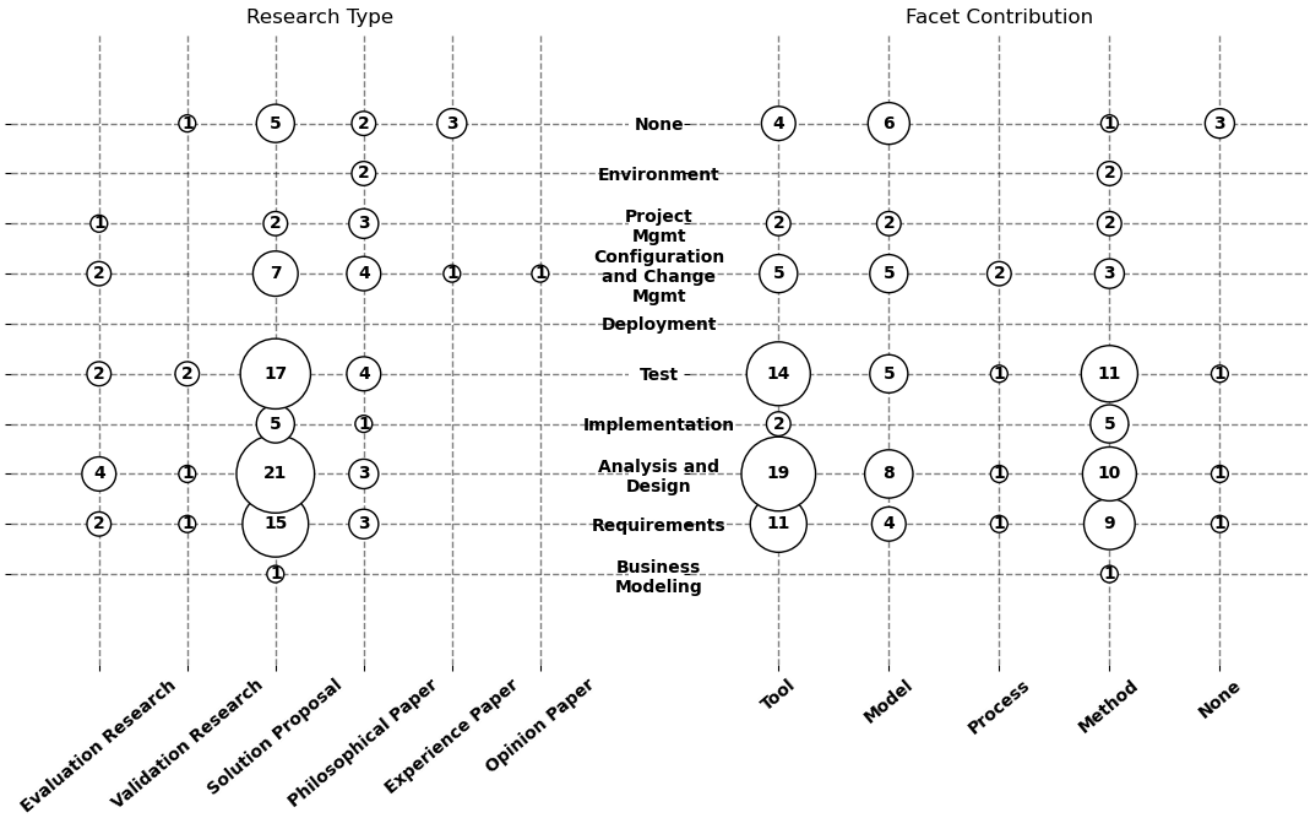


Figure 9: Bubble-plot of OSLC standard by research type and contribution facets.

the following we describe each one of the adopted terminologies for Software Engineering disciplines:

- Business Modeling: aims to describe how the product will be developed from the, taking into account the processes, roles and customers of organization.
- Requirements: the goal of this discipline is define the scope of system from stakeholder requests.
- Analysis & Design: shows how the system will be developed from models of tasks, source code or software components.
- Implementation: this discipline is composed by activities to develop source code such as classes and objects.
- Test: is devoted to ensure the product quality, including the verification and validation of assets such as requirements, diagrams, etc.
- Deployment: this discipline is focuses in activities to deliver the software to its end users.
- Configuration & Change Mgmt: includes activites to track versions, manages controlling changes, and change requests.
- Project Mgmt: is devoted to tasks involving the project management and risk management.
- Environment: is focused on activities to provides a software development environment configured and ready for work.

According to the SMS results shown in Table 12, most of the proposed solutions are to the requirements, analysis & design, test, and change & configuration management domains. All tools in the toolchain were considered part of the OSLC solution. Therefore, studies can be categorized into more than one discipline.

OSLC approaches endeavor to solve problems that are common among the papers. In the requirements domain, OSLC is proposed to establishing traceability between requirements and the remaining of the project. Besides, software requirements tools are often used in the toolchains examples as a starting point for workflow.

From Linked Data concepts, papers such as [80], [21] proposes approaches to track the history of the changes through OSLC resources shapes, including stakeholders that modified it, and tasks in which the artifacts are associated. In this sense, Change & management domain approaches such as [47], [63], [1], [64], [7], [6] are proposed to version control and task automation.

SMS results strengthen the need to integrate different domains of software development from its earliest phases. Therefore, we can mention tools for Model-Driven Engineering, which are cited in the Analysis & Design phase as solutions to minimize these integration efforts. Eclipse Lyo [25] proposes an open-source code generator for OSLC-compliant tool interfaces in the Eclipse environment which was cited by papers that were used in the development of OSLC adapters. Some works [23], [51], [8], [55], [89], [14], [15] proposes approaches to support integration or transformation of heterogeneous models such as BPMN, UML, SysML, SoaML, EMF.

Table 4: Research Groups and Evolution of Their Contributions to the Research Topic

Research Group	ID	Year	Description of the main contribution of the paper
Aichernig's Research Group	s41	2014	This paper proposes a methodology to support the transformation and validation of textual to formal requirements. OSLC consumers and providers are established to create traceability links between artifacts
Arnould's Research Group	s19	2018	This paper proposes to apply a model-based system engineering approach (MBSE) to enable integration between the different systems that support the engineering process in the Navy domain using OSLC adapters
Bergs's Research Group	s58	2010	This paper presents the implementation of OSLC specifications in some existing tools, projects, and models proposed both for Open Source Software
Buffoni's Research Group	s22	2017	This paper presents an approach to support traceability in the requirements verification results on the Modelica platform. Traceability is supported via the OSLC specification standard combined with the Git version control system
Zhang's Research Group	s54	2012	This paper a model-based approach to integrating tools based on heterogeneous tool metamodels using OSLC
	s57	2012	This paper proposes an approach to extend OSLC domains to support roles attached to artifacts
	s53	2013	This paper proposes an approach to establishing toolchains at the SaaS level. The method includes the transformation of UML, BPMN, and SoaML models to OSLC through MDA and M2T activities
	s6	2014	This paper proposes an approach to extend OSLC domains to support roles attached to artifacts
	s18	2018	This paper proposes a service-oriented toolchain to support model-based system engineering (MBSE) of aero-engines. The toolchain transform simulation data, models, and tool operations to web-based OSLC services
	s15	2019	This paper proposes an approach to support data integration and management of IoT Systems based on OSLC specification
Biehl's Research Group	s55	2012	This paper proposes an approach to semi-automatically generation of OSLC adapters through EMF metamodels representation
	s56	2012	This paper proposes a model-based approach to address both discovery and orchestration for RESTful services in the OSLC toolchains from tool adapter metamodel
	s49	2013	This paper proposes an approach to integrate web and desktop tools by combining the TIL and CORSET approaches
El-Khoury's Research Group	s8	2016	This paper proposes an open-source code generator for OSLC-compliant tool interfaces in the Eclipse environment
	s35	2016	This paper evaluates the Eclipse Lyo, a fully-automated code generator that provides OSLC interfaces for EMF-based modeling tools
	s9	2016	This paper proposes guidelines for OSLC and STEP approaches to deal with data exchange, sharing and change management in heterogeneous environments
	s29	2017	OSLC and STEP, s9
	s10	2018	This paper proposes an approach to represent the data model of robots Key Performance Indicator (KPIs) in automated warehouses through OSLC resources shapes
	s12	2019	This paper presents experiences and lessons learned in the industry in tool interoperability approaches based on Linked Data and OSLC standard
	s16	2019	This paper proposes an architecture based on Linked Data services for the integration of Cyber-Physical Systems components to exchange state updates among CPS components via OSLC
Mustafas's Research Group	s27	2017	This paper proposes an OSLC-based taxonomy to capture traceability information among software artifacts in the Requirements Engineering, Model-Driven Engineering, and Systems Engineering Domains
	s20	2018	This paper proposes a trace link taxonomy to capture traceability information among artifacts. The taxonomy employs the RDF and OSLC approaches for defining a set of properties and their values for each trace links
Loiret's Research Group	s34	2016	This paper proposes an approach to identify what, when, and how information should be integrated. In addition, it suggests integration between IoT and PLM platforms using semantic OSLC specifications
	s14	2019	This paper proposes an approach to a static code analysis toolchain that allows the link between the code and the results of the static analysis through the standardization of the data exchange format
Gallina's Research Group	s23	2017	This paper proposes an OSLC-based toolchain enabling continuous self-assessment, via the continuous semi-automatic creation of safety cases aimed at extracting information from an RDF-graph, representing an instance of interconnected domains
	s24	2017	This paper establishes guidelines and the implementation of an ISO 26262-compliant OSLC-based knowledge domain
Elaasar's Research Group	s50	2013	This paper discusses the challenges to design management and proposes the development of a server-based application that implements the OSLC's Linked Data principles
	s48	2013	This paper proposes an approach to integrate MOF-base modeling tools with other lifecycle tools using OSLC
Ayoama's Research Group	s51	2013	PROMIS
	s43	2014	This paper presents a resource-oriented service architecture for exchanging data based on OSLC. There are developed a platform for publishing and retrieving project management data in the form of RDF
Regan's Research Group	s59	2014	PAM
	s39	2015	This paper presents the development of a traceability process assessment model (PAM) to assist medical device organizations in addressing the lack of guidance. This approach uses OSLC to access resources artifacts and their relationships across all tools

Table 5: Research Groups and Evolution of Their Contributions to the Research Topic 2

Research Group	ID	Year	Description
	s44	2014	This paper presents the author's expectations about OSLC regarding the hype cycle applied to emerge technologies
Biro's Research Group	s33	2016	This paper presents an approach to improve the completeness and consistency of traceability in either homogeneous or heterogeneous tool environments through OSLC, which is based on Linked Data principle
	s4	2016	This paper proposes to replicate the development environment into a virtual space to identify incorrect or missing artifacts relationships. The OSLC standard establishes these relationships
	s26	2017	Augmented Lifecycle Space
	s28	2017	This paper proposes an approach to automatically checking bidirectional traceability and consistency of requirements using OSLC specifications
Romanovsky's Research Group	s31	2016	This paper presents a prototype of a tooling platform that integrates the process of defining requirements for a safety-critical system with formal modeling in the Event-B approach through OSLC adapters
	s37	2015	This paper proposes a reactive architecture to enhances the integration of software system lifecycle tools. The architecture uses OSLC to create a reactive middleware that informs all stakeholders about any changes in the development artifacts
Troubitsyna's Research Group	s13	2019	This paper discusses the challenges for creating an integrated environment in intelligent systems area and demonstrate how OSLC can facilitate this integration
Ryman's Research Group	s5	2013	This paper discusses the OSLC resources shape specification as the solution for the need for a constraint language in the industry
Seceleanu's Research Group	s47	2013	This paper reports the development of such an OSLC adaptor for the requirements management module of an ALM tool
	s5	2016	This paper describes an approach for integrating version control tools in the context of the embedded systems through OSLC adapters
Ellen's Research Group	s1	2013	This paper defines a guideline to systematically connect tools through an OSLC-based interoperability specification in the form of a recipe
	s46	2014	This paper presents the development of a toolchain based on the concepts of Reference Technology Platform (RTP) using OSLC as an interoperability technology
Saadtmand's Research Group	s45	2014	This paper investigates the relationship between OSLC and Systems Engineering, highlighting key concepts and keywords for defining and comparing both approaches
Siebenmarck's Research Group	s42	2014	This paper proposes the development of a tool for ALM projects visualization as graphs of the relationships established by the artifacts through OSLC's Linked Data approach
Herbst's Research Group	38	2015	This paper discusses how to integrate tools using OSLC adapters in the Smart Factory Context
	s2	2015	This paper presents an approach based on the Xtext framework to transform software requirements written in natural language into semi-formalized requirements through the lightweight integration concept based on OSLC
	s32	2016	This paper discusses lessons learned and challenges in the design and implementation of OSLC interfaces covering essential aspects of adapters, such as communication, reuse, and trace links
VanZandt's Research Group	s3	2015	This paper proposes an approach to improving the reliability of artifacts and helps decision-making in the project. OSLC is used to establishing version control, including who, when, and why the artifact was changed
Martino's Research Group	s7	2016	This paper provides an idea for creating a uniform OSLC-based semantic representation to integrating BPMN and UML models
Nardone's Research Group	s11	2020	This paper describes the development of an environment for the functional system-level testing of railway controllers, relying on OSLC to enable interoperation with existing PLM/ALM tools
Pikus's Research Group	s17	2019	This paper presents a semi-automatic end-to-end documentation system for extracting distributed lifecycle data, for storing in RDF files based on OSLC domain ontologies
Rodriguez's Research Group	s21	2018	This paper proposes an approach to artifact reuse via OSLC-based specification to share and exchange data artifact through the Linked Data principles. A new OSLC domain, OSLC KM, has been defined and implemented for knowledge management
Vallaca's Research Group	36	2016	This paper presents a case study experience with tool integration in the Aircraft industry using approaches such as OSLC, SPEM, and MBSE
	s25	2017	This paper proposes an OSLC-based approach to provide interoperability between tools used in the Model-Based design and analysis of aircraft systems
Rene's Research Group	s30	2017	This paper provides a model-based system engineering approach to enable the traceability artifacts within the context of VR/AR tools through OSLC specifications
Shahrokni's Research Group	s40	2015	This paper discusses approaches to integrate data management based on experiences in the embedded and automotive industry

Another observation is a considerable number of solutions applied to the test discipline due to the contexts in which OSLC is usually applied. As these are safety-critical systems, toolchains are composed by V&V tools.

Most of the papers are not associated with a specific software development process. However, In eight papers [58], [43], [33], [34], [19], [18], [24], [42], the V-model process was mentioned by authors as the development process present in the environment which the solution is validated.

OSLC advantages are achieved from linked data principles. Through OSLC domains specification in which artifacts are represented, data traceability, consistency and exchange are achievable. Consequently, the collaboration between different development teams is improved.

In regards to the disadvantages, it was possible to highlight: 1) it is hard to implement OSLC, either due to the lack of documentation or the high technical knowledge required; and 2) OSLC requires a larger development effort than the required by a simple web service integration.

4.2 What are the contexts of organizations using the OSLC standard? (RQ2)

Most of the studies found are contextualized in critical security environments, such as medical systems, embedded systems, the automotive industry, and the aeronautical industry, as shown in the Figure 10. These systems are characterized by focusing on verification and validation activities, such as simulations and different levels of tests. In other 13 works, the environment context is unclear. The context of organizations categories are explained as follows:

- Medicine: Systems for medical environments.
- Embedded Systems: Computer systems responsible for performing only one specific task.
- Academic: Systems development in collaboration with universities.
- Automotive: Systems development to automotive context.
- Safety-Critical: Systems that provide for the execution of V&V activities because it is not allowed to fail because it involves human lives or major financial loss.
- CPS/IoT: Environments that integrate software and hardware.
- Airplane: Development of systems for aircraft systems.
- Augmented Reality: Virtual reality (VR) systems.
- Navy: Development of systems for marine vessels.

The companies that provided the validation of OSLC specifications in the industry are large, involving a significant number of employees, such as the automaker Scania. Due to the requirements imposed for the operation of these systems, either by governmental standards or ISOs, OSLC is proposed as an alternative to establishing integrations in the data layer of the applications that form these ecosystems, especially in the test environments.

Although OSLC meets the needs of these organizational environments, the authors show that integrating environments is not a trivial task. In addition, the integration of these environments requires specialized professionals, which can increase the cost of their adoption. One of the options that companies seek is to adopt approaches such as MDD and MDE for the generation of integrating interfaces. This scenario can be an alternative for medium and small companies to adopt OSLC in their environments.

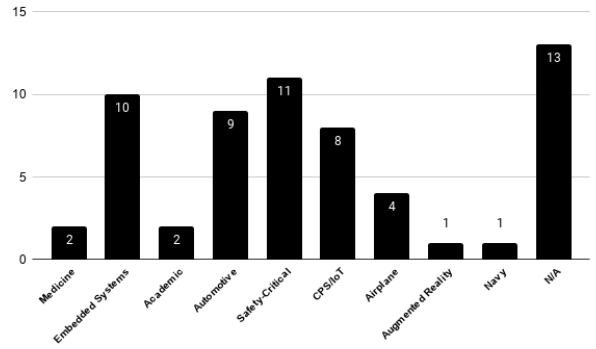


Figure 10: Safety-critical environments.

4.3 What are the most frequent research types about the OSLC standard? (RQ3)

The maturity of a research can be determined by how the study affect the practice in the area. For this reason, we adopted the categories proposed by the Wieringa study [83], mapping the research types employed by each research. Thus, the following types of studies are considered:

- Evaluation research: is any paper that presents a empirical study such as case study, field study, survey, etc.
- Validation research: is composed by studies that presents quantitative research such as experiments, simulation, prototyping, mathematical analysis, etc.
- Solution proposal: These papers presents a proof-of-concept such as a development of a small example or a sound argument.
- Philosophical paper: Any paper that proposes new conceptual approaches or discussions a new way at looking at existing approaches.
- Opinion paper: These papers discusses the author's opinions about advantages or drawbacks of research topics, how the industry should use approaches, etc.
- Experience paper: is composed by lessons learned in the industry experience such as challenges in projects, development of products, evaluation of approaches, etc.

Table 9 shows the majority of the papers classified as solution proposals (36), strengthening the idea that OSLC is implemented as a proof of concept and prototypes in real environments. Some papers suggest performing case studies, although insufficient data are presented, leading to a small number of empirical studies, including validation (3) and evaluation research (5). Among the studies, 12 philosophical papers discuss how to use OSLC to solve problems, but they do not present its implementation. There are also guidelines, opinions, and reports based on professional experiences in the industry, resulting in 5 experience papers and 1 opinion papers, increasing that OSLC is used in real contexts.

As shown in Figure 11, since the first publication on OSLC in 2010, there has been an increase in publications with its peak in 2016. However, the number of papers on OSLC until the first half of 2020 has decreased. The authors of the works found, in addition to

Table 6: Papers x Disciplines

ID	Business Modeling	Requirements	Analysis & Design	Implementation	Test	Deployment	Configuration & Change Mgmt	Project Mgmt	Enviroment	ID	Business Modeling	Requirements	Analysis & Design	Implementation	Test	Deployment	Configuration & Change Mgmt	Project Mgmt	Enviroment
s1		X	X		X					s31		X							
s2		X			X					s32									
s3							X			s33		X	X	X	X				
s4								X	X	s34									
s5							X			s35			X						
s6		X	X		X					s36		X	X		X		X	X	
s7	X	X	X	X						s37							X		
s8			X							s38		X	X		X				
s9			X		X					s39							X		
s10										s40									
s11		X	X		X					s41		X			X				
s12										s42									
s13										s43							X	X	
s14				X	X					s44									
s15										s45					X				
s16										s46		X	X		X				
s17							X			s47		X	X	X	X				
s18			X		X			X		s48			X						
s19										s49		X			X				
s20		X	X		X					s50			X				X		
s21		X	X		X					s51							X	X	
s22		X			X		X			s52							X		
s23		X	X	X	X					s53		X	X		X		X		
s24					X					s54		X	X		X				
s25			X		X					s55			X		X				
s26								X	X	s56			X						
s27			X							s57			X						
s28		X	X	X	X					s58							X		
s29			X							s59							X		
s30		X	X																

Table 7: Studies Distributed by the RUP Disciplines

Business Modeling	1
Requirements	21
Analysis & Design	29
Implementation	6
Test	25
Deployment	0
Configurantion & Change Mgmt	14
Project Mgmt	6
Environment	2
NA	11

Table 8: Studies Distributed by Application Contexts

Medicine	2
Embedded Systems	10
Academic	2
Automotive	9
Safety-Critical System	11
CPS/IOT	8
Airplane	4
Augmented Reality	1
Navy	1
NA	13

publishing their initial proposal, continue the research theme, and have worked on its evolution. It shows that the group of respondents is considered small, focusing on a few domains in the industry. It indicates that the more the area grows, new proposals lose space as the demand for applied studies increases as a way of validating these approaches.

4.4 Which are the facets of the study contribution? (RQ4)

Identify which facets of contribution are most explored in a research area is important for finding research gaps or assessing trends in the areas. Regarding the studies contribution facet, we established four categories, as seen in Table 11. These facets are:

Table 9: Research Types on OSLC

ID	Evaluation Research	Validation Research	Solution Proposal	Philosophical Paper	Experience Paper	Opinion Paper	ID	Evaluation Research	Validation Research	Solution Proposal	Philosophical Paper	Experience Paper	Opinion Paper
s1			X				s31			X			
s2			X				s32					X	
s3				X			s33				X		
s4				X			s34			X			
s5			X				s35			X			
s6			X				s36	X					
s7			X				s37			X			
s8			X				s38				X		
s9			X				s39	X					
s10		X					s40					X	
s11			X				s41			X			
s12			X		X		s42			X			
s13				X			s43			X			
s14			X				s44					X	
s15			X				s45				X		
s16			X				s46			X			
s17			X				s47			X			
s18			X				s48	X					
s19				X			s49			X			
s20	X						s50			X			
s21		X					s51				X		
s22			X				s52				X		X
s23			X				s53				X		
s24		X					s54			X			
s25			X				s55			X			
s26				X			s56			X			
s27			X				s57			X			
s28			X				s58					X	
s29	X						s59			X			
s30			X										

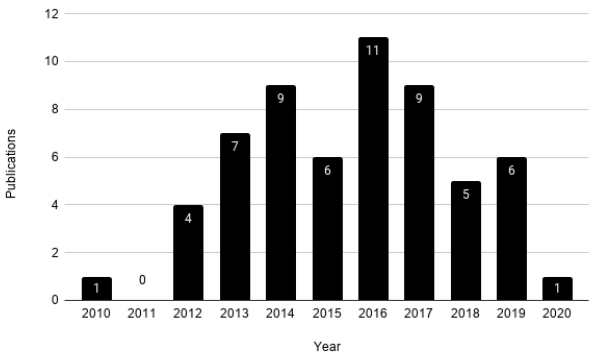


Figure 11: Publications Over the Years.

- Tool: is composed by pieces of softwares such as adaptaders, web services, software to automate activities, etc.
- Model: is a graphic representation of a activity or component of software in a standard specification.

Table 10: Study Distribution by Research Type

Evaluation Research	5
Validation Research	3
Solution Proposal	36
Philosophical Paper	12
Experience Paper	5
Opinion Paper	1

- Process: is a set of activities that leads a development of a software or the execution of project.
- Method: is composed by approaches that describes how to use a technology through the software development lifecycle.
- Metric: is a qualitative or quantitative standard used to measurement a system or a software process.

Results show that most of the papers contribute to the tools facet. This category is related to the implementation of OSLC tool adapters, OSLC Web services, or tools to visualize data artifacts. Works such as [52], [58] show toolchain examples that use OSLC adapters as an integration solution. Regarding the model category,

works such as [90], [37], [4], [34], [55] covers proposal for extending the OSLC domains specifications. Another representative topic in the model category are approaches to generating OSLC specifications in MBSE-based environments. In the examples of processes found, OSLC is used to automate activities that involve the traceability of artifacts as shown in works such as[9], [64], [65].

In another papers, methodologies or new approaches for the use of OSLC are proposed.Thus, it was possible to identify that the studies' contributions are focused on solving problems in a specific domain, such as an activity or a type of industry. It is indicative of the research gap of approaches to global solutions about OSLC, in specially planned to SE processes.

Table 11: Facet of contribution study

ID	Tool	Model	Process	Method	Metric	ID	Tool	Model	Process	Method	Metric
s1			x			s31	x				
s2	x					s32	NA				
s3		x				s33				x	
s4				x		s34	x	x			
s5	x					s35	x			x	
s6	x	x		x		s36	x				
s7				x		s37		x			
s8	x					s38	NA				
s9	x			x		s39			X		
s10		x				s40	NA				
s11		x				s41	X				
s12	x	x				s42	X				
s13		x				s43		X			
s14	x			x		s44	NA				
s15	x	x				s45				X	
s16		x				s46	X				
s17	x	x				s47	X				
s18	x					s48	X			X	
s19				x		s49	X			X	
s20		x				s50	X				
s21	x	x				s51		X			
s22				x		s52				X	
s23				x		s53				X	
s24		x				s54	X			X	
s25	x					s55	X			X	
s26				x		s56	X	X			
s27		x				s57	X	X			
s28				x		s58	X				
s29	x	x				s59			X		
s30	x			x							

Table 12: Study Distribution by Contribuition Facet

Tool	30
Model	20
Process	3
Metric	0
Method	20

5 DISCUSSIONS

OSLC specifications are becoming increasingly popular in the software industry. Over the years, challenges and practices have been reported in the literature to connect data across domains, applications and organizations. In this sense, this empirical work helps to understand the potential areas of OSLC employment based on analysis of selected papers. For instance, our results show the motivations and contributions of OSLC to support tool integration based on Software Engineering disciplines, industrial context, facets of the contribution and evaluation type of such works.

In this section, we discuss and reflect on our findings and trends in research topics. The works found focus on the integration of tools, the generation of source code to connect tools and the exploring resource shape of OSLC to connect data. The authors still look for ways to represent the artifacts and explore ontologies and extension of the OSLC domains. In addition, it was possible to realize that it is an industry consensus to integrate tools through adapters. One reason is that as tools generally do not have native OSLC support.

MDD and MDE approaches are triggered that are of interest to the industry to generate these interfaces. Over the years, it has been consolidating itself as the best alternative. Despite this, code generation is not 100% automatic and has not yet reached the level of integrating the entire environment. This scenario can be correlated with the technical challenge necessary to integrate such applications.

5.1 Establishing traceability between artifacts with OSLC

Toolchains generate heterogeneous artifacts often in the same project domain. For instance, in safety-critical systems projects, different kinds of models are used on analysisproject discipline, such as SysML, UML, and BPMN. In this sense, one of the motivations for adopting OSLC is its ability to establish relationships through linked data. These relationships are created in resource shapes, which make it viable to represent the properties of an artifact (e.g. creator, contributor, description) in XML/RDF format.

OSLC appears as an alternative for environments that need to implement traceability. Traceability is the ability to establish links (or traces) between source artifacts and target artifacts [36]. In this sense, some solutions exploit these properties to allow traceability between requirements and the remainder of the artifacts. Thus, some authors propose the extension or even a new OSLC domain to establish traceability through the toolchain. In this sense, some solutions exploit these properties to allow traceability between requirements and the remainder of the artifacts. For such, some authors propose the extension or even a new OSLC domain to establish traceability through the toolchain.

Our findings in the literature, together with the experience reports on the OSLC usage, may assume that the OSLC domains are not enough for some problems, requiring customization. This need may be associated with the vague documentation of the OSLC domains, making the need for interpretation by the developers to choose and implement the necessary attributes.

The ability to implement traceability in all environment is in line with ALM's motivations. One of the problems in achieving this objective is difficulty in maintaining data consistency. In this sense,

OSLC can be an alternative to automate ALM environments due to its ability to maintain these relationships in a structure called triple, including in addition to the artifacts, stakeholders and related activities.

Based on this, approaches to replicate and connect the entire environment, motivated by ALM approaches, were also proposed. For instance, such as the replication of the development environment for an ALM platform to enhance the services of the artifacts and maintain their consistency [46]. For this, the authors emphasize that the environment must be integrated and following the OSLC specifications. In parallel with this topic, we find works that propose the use of OSLC to establishes ontologies [57]. An ontology provides a set of classes, properties, and restrictions that can be used to represent and interchange information [61]. These approaches are defined using Web Ontology Language (OWL), which extends the XML/RDF format, in which the artifacts are represented.

5.2 Model-Driven Engineering approaches

OSLC is motivated by point-to-point integrations, and for that, it is necessary to create an interface for the exchange of data between applications defined as adapters. Adapters aim to transform data from one application to the format of the other application. The great thing about the OSLC standard is that if a tool supports OSLC, any tool integrated into that chain can exchange data. The creation of these adapters is not a trivial task, being costly for the organization. There is a need to filter the tools, the resources that will be exchanged between them, the attributes that will be represented by the OSLC, and even the relationships linked in the toolchain. For this, MDE approaches make it possible to abstract the complexity of these environments, enabling reuse.

There are approaches to generate adapters code through model representation. These approaches have already been validated in real contexts and have several positive points about the choice of model-driven approaches. In this regard, the context in which it was validated reports that MBSE and MDE approaches are conventional in these environments.

Works focus on the evaluation of Eclipse Lyo as a solution for the semi-automatic code generation of OSLC interfaces through the representation of EMF [14] [27] metamodels. These works discuss the development of adapters for analysis and design tools, motivated by the heterogeneity of the models generated by these tools. Despite this, only code skeletons are generated and need to be completed manually. These have already been validated in real contexts and have several positive points about choosing a model-based approach such as the reuse of adapter models in other tools and that it is not necessary to know the internal technology of these tools.

In addition to Eclipse Lyo, we find the work of Biehl *et. al* [16] which proposes the construction of OSLC adapters to integrate web and desktop tools, using DSL Tool Integration Language (TIL). This work is motivated by the activity of replacing desktop tools with web solutions so that these tool chains remain stable during this transition period.

Finally, there is no recipe for developing OSLC adapters. Among the common aspects in common in the related works are that the authors strictly follow the OSLC specifications, seeking to map the attributes of the artifacts shared with the OSLC domains and although it is a viable alternative for the construction of adapters, its use in the industry indicates that it is not a trivial activity, needing a domain knowledge specialist. In addition, the solutions proposed by the works found in the literature indicate that there is an interest in the industry to improve the code generation of OSLC adapters through the representation of models.

5.3 Internet of things and OSLC approaches

With the rise of the internet of things (IoT) and with the world increasingly connected, it is mandatory to supply the need to connect and maintain the consistency and traceability of this data of such systems. The works report that OSLC is exploited for the integration of manufacturing systems, intelligent installations, and 3D modeling. These environments are characterized by handling a large volume of data, requiring hardware and software to work in harmony. One of the points explored is the ability to standardize data formats and the communication between components through OSLC-based service architectures.

5.4 V&V environments

Our results show that OSLC is mostly used in safety-critical systems. Some examples are the contexts of the automotive industry, the airline industry and the development of embedded systems. These environments are characterized by the need for verification and validation activities involving simulations and formalization activities. For example, Author [2] proposes the integration of tools in test environments for the formalization of requirements and Nardone *et al.* [58] describes the development of an environment for functional testing at the level of system for rail controllers integrated with OSLC adapters.

Other authors explore OSLC to integrate environments based on ISO 26262, used in the embedded systems industry. The software development in this context follows the model-V process, which is based on activities of verification and validation of the software artifacts produced during the process.

6 THREATS TO VALIDITY

Construct Validity: We followed a well-defined protocol widely used in the Software Engineering area. To ensure the data extraction, the classification scheme was reviewed and discussed by a second author. Our research does not have quality assessments; therefore, it is not possible to evaluate the relevance of primary studies to answer each research question. Once the quality assessments are rarely found in mapping studies and are more appropriate for comparative studies, such as Systematic Literature Reviews, this construct validity threat is not considered critical to answer our elaborated research questions. Another threat is that the categories chosen to answer in which domains the OSLC is employed may be considered generic. To mitigate this bias, the categories were chosen from the Rational Unified Process base of knowledge, allowing us to achieve an overview of a well-known research theme for Software Engineering research.

Internal Validity: Our conclusions are drawn based on well-defined research questions. Besides, an ongoing concern throughout the study was to ensure that selected primary studies were consistently identified and analyzed according to inclusion and exclusion

criteria. Hence, it was carefully invested in a huge effort to filter primary studies. In this sense, some studies were not associated with at least one exclusion criteria but also did not fulfill any inclusion criteria. Thus, to remove this bias, these studies were not selected.

External Validity: Systematic Mapping Studies include more papers than other empirical studies such as Systematic Literature Reviews. However, the statistical representativeness of our findings may be a threat due to a significant number of papers. This bias was handled by improving the search strings by researchers, including 59 papers, which characterize a good statistical power, and the adaptation of the sequence for each of the selected databases.

Conclusion validity: Threats to SMS results can occur in the classification phase, which may be influenced by the know-how author's bias. In other words, mapping studies can suffer from this validity when the research has an *ad-hoc* classification of existing work or when a unique reviewer provides the classifications, which may affect the tabular representation of contributions. To surpass this bias, we adopted nomenclatures from well established Software Engineering disciplines and Computer science application domains. This threat was also mitigated, with a second author actively participating in the whole execution process.

7 RELATED WORKS

OSLC, despite being motivated by approaches that seek the end-to-end integration of the software development environment, such as DevOps, ALM, PLM and IoT, as seen in our work, solutions that achieve this goal in the industry are not developed. In parallel with our research topic, we have the theme of continuous software engineering, which seeks to automate the entire software development process, which includes the integration of tools. In this context, Gerald et al. [30] argues in their work that due to the growing popularity of agile methodologies by companies recently, organizations have been looking for alternatives to achieve the continuous integration of the environment. Despite this, the authors argue that due to environmental complexities, activities in the development, business modeling and deploy phases are disconnected from each other. Still, the authors illustrate the disconnected scenario of organizations, including at the tooling level. Another point of this work is that the authors show that there is an interest by the industry about the DevOps approach, which may be a topic of interest to explore about the OSLC, since in our mapping there were no approaches related to DevOps. In their conclusion, the authors leave some questions that open the opportunity for OSLC in topics of Business strategy, development and operations, highlighting "How can hardware and software co-development follow a continuous software engineering approach?" and "What architectural solutions offer the highest degree of flexibility to facilitate continuous evolution?". Based on this, OSLC, through a common service architecture, can meet this need as it provides benefits such as data traceability, flexibility in the reuse of assets (tools), low coupling and data reliability.

This scenario leads us to another important theme for the evolution of the research, which is the opportunistic reuse, an approach in which new software systems are built from assets that were not originally designed to be combined, such as the implementation of a tool chains. The work of Makitalo *et al.* [**?**] conducted a survey with industry professionals on how developers select these assets. After analyzing the results, the authors confirmed some premises that opportunistic reuse is applied manually and *ad hoc* by the developers through their personal experience. This leads to problems such as "slowness" in the asset selection process due to the different versions and dependencies of those assets. That also leads to other results of the survey: the choice of these assets are "tied" to an architecture, because the non-standardization of these assets can result in architectural issues. When we talk about tooling, this scenario illustrates the opportunity to use a common service-based architecture to solve these problems. The problem of non-standardization of assets has already been addressed in the work of [56], which carried out a systematic review of the literature pointing out the need for traceability in heterogeneous systems emphasizing the need for a specification to link artifacts in heterogeneous systems.

Finally, the work of [54] discusses opportunities for opportunistic reuse of software and trends for works that explore this area of research, such as the tooling apparatus to support such practice. One of the "Call for Action" opportunities of this work is in the development of OSLC solutions that assists in the monitoring of selected components/assets to achieve traceability and consistency, including the evolution of these components. With that in mind, it would be possible to view the dependencies and the version history of these assets, collaborating for the adoption of opportunistic reuse.

Our systematic literature review is complementary to the aforementioned works, but it is also singular in the literature of the area. First, we follow a replicable protocol that can be implemented over the years to identify new contributions in the area. Also, we categorize contributions in chronological order, research groups and maturity levels. This factor is especially important for the academic community, as it provides an overview of the state of the art and practice, on the areas explored, the shortcomings to adopt the OSLC and also the unexplored areas.

Besides, this work can serve for decision making in the development of integration solutions in the industry. It includes decision making on the adoption of techniques ranging from the decision to implement OSLC in certain domains, choosing the best way to implement OSLC connector interfaces and decide if the contexts of the companies where OSLC was explored fit the new reality.

8 CONCLUSIONS

Through a systematic mapping literature review, this paper characterized the OSLC usage for tool integration in the context application lifecycle management for Software Engineering. This review scopes four research questions about the state-of-the-art and practice in tool integration, searching for ALM phases, most used tools, processes and roles, development contexts, contribution facet, and evaluation, as well as advantages and disadvantages of OSLC.

OSLC is motivated for integration issues from high complexity industrial contexts. In this sense, the proposed integration solutions by the authors are generally designed for specific domains of computer engineering software applications. For example, most of the studies are devoted to Embedded Systems and other application contexts from computer engineering research, such as Industrial

Automation, Automobile, and others. Does this mean that the research field needs more empirical evidences?

Our work also investigates the roles involved in the ALM lifecycle, finding reports suggesting that these software development stakeholders are tool experts. This need for experts is classified by researchers as a big disadvantage for tool integration with the OSLC standard. We also found articles discussing the experience of professionals using OSLC in the industry, but there is not a significant number of empirical studies proving that knowledge.

Although OSLC proposes end-to-end integration of the entire ALM lifecycle, studies show that existing toolchains are, at their largest sample, adopted in the early stages of the software project, and also those involving verification and validation activities. Once many current research papers are devoted to continuous deployment and delivery in Software Engineering research [70], we also missed studies about tool integration devoted to the Deployment phase. Thus, this characterizes as a research gap: how integration performed with OSLC considers the software projects that adopt continuous integration platforms?

Among the main motivations for building OSLC-based solutions is the possibility of establishing traceability links between software tools. In this context, OSLC software environments can achieve characteristics that some approaches in which Software Engineering is applied recommend, as in some levels of Capability Maturity Model Integration (CMMI) [75].

The main OSLC advantages are related to linked data. It involves not only tool adapters for point-to-point integration, but also proposed approaches to promote tool replacement in the toolchain, including modification of OSLC domain specifications and solutions for automated activities for tool integration. All of these elements can be satisfied through approaches such as MDE, which by a common representation, can provide the entire device that allows the automation of activities and integration of the whole software development environment.

Although OSLC has evidence of use in industry, there are few empirical studies of type "Evaluation research". The industry still must visualize the advantages of using OSLC. The OSLC community has focused on developing solutions for safety-critical systems even though data exchange between tools is a common problem in other domains. This means a focus on contributions for tool integration in "Requirements", "Analysis an Design" and "Test". The full potential from OSLC is not yet explored in Software Engineering disciplines such as the "Business Modeling", which is essential for the development of specific domains such as Information Systems and Enterprise Applications, and "Deployment", which is usually associated with automated processes for Continuous Integration and Continuous Delivery.

Therefore, there is a research gap for applied studies in terms of completeness for supporting software development lifecycle phases. Meanwhile, we also provided a panoramic view regarding whether previous expectations from research gaps claiming the need for software process automation through systems-of-systems integration [20, 32] are materialized through the OSLC standard. Likewise, the answer to the four research questions allowed us to infer a new one in the conclusion: whether software development processes are fully automated through the surveyed studies?

The answer to this question is No, because the solutions proposed by the OSLC studies are applied to specific problems. We have identified that OSLC is used in core systems development activities, including a defined toolchain for the system engineering research, rather than implementing a systems-of-systems integration devoted to Software Engineering integration issues. This is paradoxical since OSLC was proposed for SE integration issues. Thereby, in the context of systems supporting Software Engineering tasks, studies are not considering application for the whole software development process through integration of many lifecycle phases.

Finally, in the contribution facet analysis, one can observe a few number of works discussing the relevance of OSLC to support software processes. This suggests that Software Engineering tools has few participation within the tool integration research scoping the OSLC specification. That is, there is a research gap by applied studies that seek the automation process as a whole, i.e., for new empirical studies non restricted to integrate tools in just one phase of the application development lifecycle.

REFERENCES

[1] David Adjepon-Yamoah, Alexander Romanovsky, and Alexei Iliasov. 2015. A Reactive Architecture for Cloud-based System Engineering. In *Proceedings of the 2015 International Conference on Software and System Process (ICSSP 2015)*. ACM, New York, NY, USA, 77–81. https://doi.org/10.1145/2785592.2785611

[2] B. K. Aichernig, K. Hörmaier, F. Lorber, D. Nickovic, R. Schlick, D. Simoneau, and S. Tiran. 2014. Integration of Requirements Engineering and Test-Case Generation via OSLC. In *2014 14th International Conference on Quality Software*. 117–126. https://doi.org/10.1109/QSIC.2014.13

[3] Bob Aiello. 2016. *Agile Application Lifecycle Management: Using DevOps to Drive Process Improvement*. Addison-Wesley Professional. https://www.xarg.org/ref/a/0321774108/

[4] J.M. Alvarez-Rodríguez, R. Mendieta, J. Vara, A. Fraga, and J. Llorens. 2018. Enabling system artefact exchange and selection through a linked data layer. *Journal of Universal Computer Science* 24, 11 (2018), 1536–1560. https://doi.org/10.3217/jucs-024-11-1536 cited By 1.

[5] AMS, 2014 2014. Asset Management Specification. Av. at <http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/>.

[6] M. Aoyama, K. Yabuta, T. Kamimura, S. Inomata, T. Chiba, T. Niwa, and K. Sakata. 2013. PROMIS: A Management Platform for Software Supply Networks Based on the Linked Data and OSLC. In *2013 IEEE 37th Annual Computer Software and Applications Conference*. 214–219. https://doi.org/10.1109/COMPSAC.2013.36

[7] M. Aoyama, K. Yabuta, T. Kamimura, S. Inomata, T. Chiba, T. Niwa, and K. Sakata. 2014. A Resource-Oriented Services Platform for Managing Software Supply Chains and Its Experience. In *2014 IEEE International Conference on Web Services*. 598–605. https://doi.org/10.1109/ICWS.2014.89

[8] V. Arnould. 2018. Using model-driven approach for engineering the System Engineering System. In *2018 13th Annual Conference on System of Systems Engineering (SoSE)*. 608–614. https://doi.org/10.1109/SYSOSE.2018.8428769

[9] A. Baumgart and C. Ellen. 2014. A recipe for tool interoperability. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 300–308.

[10] A. Berezovskyi, J. El-khoury, and E. Fersman. 2019. Linked Data Architecture for Plan Execution in Distributed CPS. In *2019 IEEE International Conference on Industrial Technology (ICIT)*. 1393–1399.

[11] O. Berger, V. Vlasceanu, C. Bac, Q.V. Dang, and S. Lauriere. 2010. Weaving a semantic web across OSS repositories: Unleashing a new potential for academia and practice. *International Journal of Open Source Software and Processes* 2, 2 (2010), 29–40. https://doi.org/10.4018/jossp.2010040103 cited By 3.

[12] Matthias Biehl, Jad El-Khoury, Frédéric Loiret, and Martin Törngren. 2011. A Domain Specific Language for Generating Tool Integration Solutions. In *4th Workshop on Model-Driven Tool & Process Integration (MDTPI)*.

[13] Matthias Biehl, Jad El-Khoury, Frédéric Loiret, and Martin Törngren. 2014. On the modeling and generation of service-oriented tool chains. *Software & Systems Modeling* 13, 2 (2014), 461–480.

[14] M. Biehl, J. El-Khoury, and M. Törngren. 2012. High-Level Specification and Code Generation for Service-Oriented Tool Adapters. In *2012 12th International Conference on Computational Science and Its Applications*. 35–42. https://doi.org/10.1109/ICCSA.2012.16

[15] Matthias Biehl, Wenqing Gu, and Frédéric Loiret. 2012. Model-Based Service Discovery and Orchestration for OSLC Services in Tool Chains. In *Web Engineering*, Marco Brambilla, Takehiro Tokuda, and Robert Tolksdorf (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 283–290.

[16] M. Biehl, J. D. Sosa, M. Törngren, and O. Díaz. 2013. Efficient Construction of Presentation Integration for Web-Based and Desktop Development Tools. In *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. 697–702. https://doi.org/10.1109/COMPSACW.2013.123

[17] Miklos Biro. 2014. Open Services for Software Process Compliance Engineering. In *SOFSEM 2014: Theory and Practice of Computer Science*, Viliam Geffert, Bart Preneel, Branislav Rovan, Július Štuller, and A. Min Tjoa (Eds.). Springer International Publishing, Cham, 1–6.

[18] Miklós Biró, József Klespitz, Johannes Gmeiner, Christa Illibauer, and Levente Kovács. 2016. Towards Automated Traceability Assessment through Augmented Lifecycle Space. In *Systems, Software and Services Process Improvement*, Christian Kreiner, Rory V. O'Connor, Alexander Poth, and Richard Messnarz (Eds.). Springer International Publishing, Cham, 94–105.

[19] Miklós Biró, Felix Kossak, József Klespitz, and Levente Kovács. 2017. Graceful Integration of Process Capability Improvement, Formal Modeling and Web Technology for Traceability. In *Systems, Software and Services Process Improvement*, Jakub Stolfa, Svatopluk Stolfa, Rory V. O'Connor, and Richard Messnarz (Eds.). Springer International Publishing, Cham, 381–398.

[20] Barry Boehm. 2006. A View of 20th and 21st Century Software Engineering. In *28th International Conference on Software Engineering (ICSE '06)*. 12–29.

[21] Lena Buffoni, Adrian Pop, and Alachew Mengist. 2017. Traceability and Impact Analysis in Requirement Verification. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT '17)*. ACM, New York, NY, USA, 95–98. https://doi.org/10.1145/3158191.3158207

[22] J. Chen, Z. Hu, J. Lu, H. Zhang, S. Huang, and M. Törngren. 2019. An Open Source Lifecycle Collaboration Approach Supporting Internet of Things System Development. In *2019 14th Annual Conference System of Systems Engineering (SoSE)*. 63–68.

[23] B. d. Martino, A. Esposito, S. Nacchia, and S. A. Maisto. 2016. Towards a Uniform Semantic Representation of Business Processes, UML Artefacts and Software Assets. In *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*. 543–548. https://doi.org/10.1109/CISIS.2016.97

[24] B. Di Giandomenico, C. Pessa, L. Valacca, E. Valfrè, and I. Viglietti. 2016. Leonardo-Finmeccanica - Aircraft division needs for integrated systems engineering: The crystal user experience. *CEUR Workshop Proceedings* 1728 (2016), 95–102. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84999274128&partnerID=40&md5=a8be9e6fb6c55ab0a075a85ea9110d59 cited By 0.

[25] Jad El-khoury. 2016. Lyo code generator: A model-based code generator for the development of OSLC-compliant tool interfaces. *SoftwareX* 5 (2016), 190 – 194. https://doi.org/10.1016/j.softx.2016.08.004

[26] Jad El-khoury, Andrii Berezovskyi, and Mattias Nyberg. 2019. An industrial evaluation of data access techniques for the interoperability of engineering software tools. *Journal of Industrial Information Integration* (2019). https://doi.org/10.1016/j.jii.2019.04.004

[27] Jad El-Khoury, Cecilia Ekelin, and Christian Ekholm. 2016. Supporting the Linked Data Approach to Maintain Coherence Across Rich EMF Models. In *Modelling Foundations and Applications*, Andrzej Wąsowski and Henrik Lönn (Eds.). Springer International Publishing, Cham, 36–47.

[28] Maged Elaasar and James Conallen. 2013. Design Management: A Collaborative Design Solution. In *Modelling Foundations and Applications*, Pieter Van Gorp, Tom Ritter, and Louis M. Rose (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 165–178.

[29] Maged Elaasar and Adam Neal. 2013. Integrating Modeling Tools in the Development Lifecycle with OSLC: A Case Study. In *Model-Driven Engineering Languages and Systems*, Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 154–169.

[30] Brian Fitzgerald and Klaas-Jan Stol. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123 (2017), 176 – 189. https://doi.org/10.1016/j.jss.2015.06.063

[31] R. Z. Frantz, A. M. Reina-Quintero, and R. Corchuelo. 2011. A DOMAIN-SPECIFIC LANGUAGE TO DESIGN ENTERPRISE APPLICATION INTEGRATION SOLUTIONS. *International Journal of Cooperative Information Systems* 20, 2 (2011), 143–176. https://doi.org/10.1142/S0218843011002225

[32] Alfonso Fuggetta and Elisabetta Di Nitto. 2014. Software Process. In *36th International Conference on Software Engineering (ICSE '14)*. 1–12.

[33] B. Gallina and M. Nyberg. 2017. Pioneering the Creation of ISO 26262-Compliant OSLC-Based Safety Cases. In *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 325–330. https://doi.org/10.1109/ISSREW.2017.41

[34] B. Gallina, K. Padira, and M. Nyberg. 2016. Towards an ISO 26262-compliant OSLC-based Tool Chain Enabling Continuous Self-Assessment. In *2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC)*. 199–204. https://doi.org/10.1109/QUATIC.2016.050

[35] Pankaj K. Garg and Mehdi Jazayeri. 1995. *Process-Centered Software Engineering Environments*. IEEE Computer Society Press, Washington, DC, USA.

[36] Orlena Gotel and Patrick Mäder. 2012. *Acquiring Tool Support for Traceability*. Springer London, London, 43–68. https://doi.org/10.1007/978-1-4471-2239-5_3

[37] Didem Gürdür, Aneta [Vulgarakis Feljan], Jad El-khoury, Swarup [Kumar Mohalik], Ramamurthy Badrinath, Anusha [Pradeep Mujumdar], and Elena Fersman. 2018. Knowledge Representation of Cyber-physical Systems for Monitoring Purpose. *Procedia CIRP* 72 (2018), 468 – 473. https://doi.org/10.1016/j.procir.2018.03.018 51st CIRP Conference on Manufacturing Systems.

[38] Gregor Hohpe. 2003. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional. https://www.xarg.org/ref/a/0321200683/

[39] Ren Hong-min, Liu Jin, and Zhang Jing-zhou. 2010. Software asset repository open framework supporting customizable faceted classification. In *IEEE International Conference on Software Engineering and Service Sciences (ICSESS), 16-18 July, 2010*. 1–4.

[40] A. Iliasov, A. Romanovsky, E. Troubitsyna, and L. Laibinis. 2016. Formalisation-Driven Development of Safety-Critical Systems. In *2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE)*. 165–172. https://doi.org/10.1109/HASE.2016.35

[41] O. Kacimi, C. Ellen, M. Oertel, and D. Sojka. 2014. Creating a reference technology platform: Performing model-based safety analysis in a heterogeneous development environment. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 645–656.

[42] C. Kaiser and B. Herbst. 2015. Smart engineering for smart factories: How OSLC could enable plug & play tool integration. *Mensch und Computer 2015 - Workshop* (2015), 269–280. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029411526&partnerID=40&md5=a32578cbd9ca9de35b4b027fa454271b cited By 0.

[43] M. Kern, F. Erata, M. Iser, C. Sinz, F. Loiret, S. Otten, and E. Sax. 2019. Integrating Static Code Analysis Toolchains. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 523–528.

[44] Anneke Kleppe, Jos Warmer, and Wim Bast. 2003. MDA Explained: The Model Driven Architecture: Practice and Promise.

[45] J. Klespitz, M. Bíró, and L. Kovács. 2016. Augmented Lifecycle Space for traceability and consistency enhancement. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 003973–003977. https://doi.org/10.1109/SMC.2016.7844854

[46] J. Klespitz, M. Bíró, and L. Kovács. 2016. Enhanced traceability and consistency with Augmented Lifecycle Space. In *2016 IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES)*. 207–212. https://doi.org/10.1109/INES.2016.7555121

[47] L. Lednicki, G. Sapienza, M. E. Johansson, T. Seceleanu, and D. Hallmans. 2016. Integrating Version Control in a Standardized Service-Oriented Tool Chain. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 323–328. https://doi.org/10.1109/COMPSAC.2016.141

[48] Andrea Leitner, Beate Herbst, and Roland Mathijssen. 2016. Lessons Learned from Tool Integration with OSLC. In *Information and Software Technologies*, Giedre Dregvaite and Robertas Damasevicius (Eds.). Springer International Publishing, Cham, 242–254.

[49] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2014. Assessing the State-of-Practice of Model-Based Engineering in the Embedded Systems Domain. In *Model-Driven Engineering Languages and Systems (MODELS'14)*. 166–182.

[50] Linked Data 2006. Linked Data Web Page. https://www.w3.org/DesignIssues/LinkedData.html Accessed at April 2020.

[51] J. Lu, J. W ang, D. Chen, J. Wang, and M. TöRngren. 2018. A Service-Oriented Tool-Chain for Model-Based Systems Engineering of Aero-Engines. *IEEE Access* 6 (2018), 50443–50458. https://doi.org/10.1109/ACCESS.2018.2868055

[52] N. Marko, A. Leitner, B. Herbst, and A. Wallner. 2015. Combining Xtext and OSLC for Integrated Model-Based Requirements Engineering. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. 143–150. https://doi.org/10.1109/SEAA.2015.11

[53] R. Matinnejad and R. Ramsin. 2012. An Analytical Review of Process-Centered Software Engineering Environments. In *Engineering of Computer Based Systems (ECBS)*. 64–73.

[54] Tommi Mikkonen and Antero Taivalsaari. 2019. Software Reuse in the Era of Opportunistic Design. *IEEE Software* 36, 3 (2019), 105–111. https://doi.org/10.1109/MS.2018.2884883

[55] N. Mustafa and Y. Labiche. 2017. Employing linked data in building a trace links taxonomy. *ICSOFT 2017 - Proceedings of the 12th International Conference on Software Technologies* (2017), 186–198. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029284603&partnerID=40&md5=14f892da98028ebb6797e10f1f106fc3 cited By 2.

[56] N. Mustafa and Y. Labiche. 2017. The Need for Traceability in Heterogeneous Systems: A Systematic Literature Review. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 305–310.

[57] Nasser Mustafa and Yvan Labiche. 2018. Using Semantic Web to Establish Traceability Links Between Heterogeneous Artifacts. In *Software Technologies*, Enrique Cabello, Jorge Cardoso, Leszek A. Maciaszek, and Marten van Sinderen (Eds.). Springer International Publishing, Cham, 91–113.

[58] Roberto Nardone, Stefano Marrone, Ugo Gentile, Aniello Amato, Gregorio Barberio, Massimo Benerecetti, Renato [De Guglielmo], Beniamino [Di Martino], Nicola Mazzocca, Adriano Peron, Gaetano Pisani, Luigi Velardi, and Valeria Vittorini. 2020. An OSLC-based environment for system-level functional testing of ERTMS/ETCS controllers. *Journal of Systems and Software* 161 (2020), 110478. https://doi.org/10.1016/j.jss.2019.110478

[59] OSLC. 2017. *OSLC Primer Web Page.* Technical Report. Open Services for Lifecycle Collaboration. open-services.net/resources/tutorials/oslc-primer/

[60] OSLC 2020. Open Services for Lifecycle Collaboration Primer Web Page. https://open-services.net/ Accessed at September 2017.

[61] OWL 2020. Web Ontology Language (OWL). Av. at <https://www.w3.org/OWL/>.

[62] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1 – 18. https://doi.org/10.1016/j.infsof.2015.03.007

[63] Yevgen Pikus, Norbert Weissenberg, Bernhard Holtkamp, and Boris Otto. 2019. Semi-automatic Ontology-driven Development Documentation: Generating Documents from RDF Data and DITA Templates. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*. ACM, New York, NY, USA, 2293–2302. https://doi.org/10.1145/3297280.3297508

[64] G. Regan, M. Biro, D. Flood, and F. McCaffery. 2015. Assessing traceability - Practical experiences and lessons learned. *Journal of Software: Evolution and Process* 27, 8 (2015), 591–601. https://doi.org/10.1002/smr.1728 cited By 6.

[65] Gilbert Regan, Miklos Biro, Fergal Mc Caffery, Kevin Mc Daid, and Derek Flood. 2014. A Traceability Process Assessment Model for the Medical Device Domain. In *Systems, Software and Services Process Improvement*, Béatrix Barafort, Rory V. O'Connor, Alexander Poth, and Richard Messnarz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 206–216.

[66] E. René and E. Martin. 2017. OSLC based approach for product appearance structuring. *Proceedings of the International Conference on Engineering Design, ICED* 4, DS87-4 (2017), 259–266. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85029763443&partnerID=40&md5=7962fc5cda03cbd3906d735534599e77 cited By 0.

[67] A.G. Ryman, A.J. Le Hors, and S. Speicher. 2013. OSLC resource shape a language for defining constraints on linked data. *CEUR Workshop Proceedings* 996 (2013). https://www.scopus.com/inward/record.uri?eid=2-s2.0-84916627887&partnerID=40&md5=714ac7f0d20d83c7e56598ec8356022c cited By 12.

[68] M. Saadatmand and A. Bucaioni. 2014. OSLC Tool Integration and Systems Engineering – The Relationship between the Two Worlds. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. 93–101. https://doi.org/10.1109/SEAA.2014.64

[69] Carey Schwaber et al. 2006. The changing face of application life-cycle management. *Forrester Research* 18 (2006).

[70] M. Shahin, M. Ali Babar, and L. Zhu. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* 5 (2017), 3909–3943.

[71] A. Shahrokni and J. Söderberg. 2015. Beyond information silos challenges in integrating industrial model-based data. *CEUR Workshop Proceedings* 1406 (2015), 63–72. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84938604236&partnerID=40&md5=d8ba0f35c028cb968d02e7e6f15889c8 cited By 2.

[72] Navid Shariatzadeh, Didem Gurdur, Jad El-Khoury, Lars Lindberg, and Gunilla Sivard. 2016. Using Linked Data with Information Standards for Interoperability in Production Engineering. *Procedia CIRP* 41 (2016), 502 – 507. https://doi.org/10.1016/j.procir.2015.12.142 Research and Innovation in Manufacturing: Key Enabling Technologies for the Factories of the Future - Proceedings of the 48th CIRP Conference on Manufacturing Systems.

[73] N. Shariatzadeh, Thomas Lundholm, Lars Lindberg, and Gunilla Sivard. 2016. Integration of Digital Factory with Smart Factory Based on Internet of Things. *Procedia CIRP* 50 (12 2016), 512–517. https://doi.org/10.1016/j.procir.2016.05.050

[74] Oliver Siebenmarck. 2014. Visualizing cross-tool ALM projects as graphs with the Open Service for Lifecycle Collaboration. In *Software Engineering 2014*, Wilhelm Hasselbring and Nils Christian Ehmke (Eds.). Gesellschaft für Informatik, Bonn, 125–129.

[75] CMMI Product Team. 2010. *CMMI for Development, Version 1.3.* Technical Report CMU/SEI-2010-TR-033. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661

[76] Ian Thomas and Brian A. Nejmeh. 1992. Definitions of Tool Integration for Environments. *IEEE Softw.* 9, 2 (March 1992), 29–35. https://doi.org/10.1109/52.120599

[77] E. Troubitsyna. 2019. Multi-Concern Integrated Engineering of Dependable Intelligent Systems. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. 710–715.

[78] Eray Tüzün, Bedir Tekinerdogan, Yagup Macit, and Kürşat İnce. 2019. Adopting integrated application lifecycle management within a large-scale software company: An action research approach. *Journal of Systems and Software* 149 (2019), 63 – 82. https://doi.org/10.1016/j.jss.2018.11.021

[79] I. Vagliano, D. Ferretto, E. Brusa, M. Morisio, and L. Valacca. 2017. Tool Integration in the Aerospace Domain: A Case Study. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 731–736. https://doi.org/10.1109/COMPSAC.2017.241

[80] L. VanZandt. 2015. Enabling rational decision making with provenance-annotated OSLC relationships. In *2015 IEEE International Symposium on Systems Engineering (ISSE)*. 346–352. https://doi.org/10.1109/SysEng.2015.7302780

[81] Anthony I. Wasserman. 1990. Tool integration in software engineering environments. In *Software Engineering Environments*, Fred Long (Ed.). Lecture Notes in Computer Science, Vol. 467. Springer Berlin Heidelberg, 137–149.

[82] M. N. Wicks and R. G. Dewar. 2007. Controversy Corner: A new research agenda for tool integration. *J. Syst. Softw.* 80, 9 (Sept. 2007), 1569–1585.

[83] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. 2005. Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion. *Requir. Eng.* 11, 1 (Dec. 2005), 102–107. https://doi.org/10.1007/s00766-005-0021-6

[84] R. Wolvers and T. Seceleanu. 2013. Embedded Systems Design Flows: Integrating Requirements Authoring and Design Tools. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. 244–251. https://doi.org/10.1109/SEAA.2013.63

[85] N.S. Zadeh, L. Lindberg, J. El-Khoury, and G. Sivard. 2017. Service Oriented Integration of Distributed Heterogeneous IT Systems in Production Engineering Using Information Standards and Linked Data. *Modelling and Simulation in Engineering* 2017 (2017). https://doi.org/10.1155/2017/9814179 cited By 1.

[86] Betty Zakheim. 2017. How Difficult Can It Be to Integrate Software Development Tools? The Hard Truth. *InfoQ* (January 2017). https://www.infoq.com/articles/tool-integration-hard-truth

[87] W. Zhang, V. Leildé, B. Møller-Pedersen, J. Champeau, and C. Guychard. 2012. Towards Tool Integration through Artifacts and Roles. In *2012 19th Asia-Pacific Software Engineering Conference*, Vol. 1. 603–613. https://doi.org/10.1109/APSEC.2012.45

[88] Weiqing Zhang and Birger Moller-Pedersen. 2014. Modeling of tool integration resources with OSLC support. In *Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on*. 99–110.

[89] W. Zhang and B. Møller-Pedersen. 2013. Establishing Tool Chains Above the Service Cloud with Integration Models. In *2013 IEEE 20th International Conference on Web Services*. 372–379. https://doi.org/10.1109/ICWS.2013.57

[90] W. Zhang and B. Møller-Pedersen. 2014. Modeling of tool integration resources with OSLC support. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 99–110.

[91] W. Zhang, B. Møller-Pedersen, and M. Biehl. 2012. A light-weight tool integration approach: From a tool integration model to OSLC integration services. *ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends* (2012), 137–146. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84868691539&partnerID=40&md5=ff30ae7cc6914be880e2eb77a11abe8a cited By 8.

APÊNDICE B – RELATO DE EXPERIÊNCIA

Exploring Tool Integration for Continuous Software Engineering with the OSLC and Eclipse Lyo

ABSTRACT

Software tools are used to support teams throughout the software development lifecycle. Yet, fully automated integrated environments are rarelly observed in the industry. In this sense, many approaches have been proposed for Continuous Software Engineering (CSE), in special those integrating tools with the Open Services for Lifecycle Collaboration (OSLC) standard. OSLC is motivated by point-to-point integrations through services. Establishing OSLC toolchains need from software engineers the know how to create tool interfaces for data exchange interoperated through adapters. This is not easy and requires well defined steps that help software engineers towards CSE automation. This paper reports on our experience through an exploratory study on a technology for modeling integration solutions, namelly Eclipse Lyo, and the respective generation of OSLC interfaces and adapters. As a result, we derived well defined steps in a metodology supporting the OSLC toolchain development process. The results are interesting and suggest that Eclipse Lyo provides some benefits to software engineers when configuring integration solutions in CSE contexts.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

tool integration, open services for lifecycle collaboration, OSLC, model-driven development

ACM Reference Format:

. 2018. Exploring Tool Integration for Continuous Software Engineering with the OSLC and Eclipse Lyo. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUÇÃO

Ferramentas de software são utilizadas para apoiar profissionais na execução de atividades ao longo do ciclo de vida do software. Essas ferramentas possuem diferentes funcionalidades, configurações, fabricantes e são desenvolvidas sem o suporte nativo para serem integradas com outras ferramentas. Além disso, manipulam artefatos (requisitos de software, modelos, código-fonte, casos de teste, etc.) produzidos em diferentes fases do desenvolvimento. Este cenário contribui para que ambientes integrados de ponta a ponta de forma totalmente automatizada sejam raros na indústria [31].

Organizações de software obtêm novas ferramentas ou desenvolvem suas soluções de software ao longo dos anos, resultando em um ecossistema de software heterogêneo [23]. Isso demanda que as organizações adotem abordagens para melhoria dos processos de software e suporte ferramental que permitam a automação do processo de desenvolvimento por meio da integração de ferramentas.

Entre as alternativas está o *Application Lifecycle Management (ALM)*, o qual adota a ideia de integração ponta a ponta, evitando silos de informações devido a características como rastreabilidade, visibilidade e automação de processos [28]. Para reduzir os problemas relacionados à integração de artefatos de software ao longo de várias fases do ALM, padrões de representação para modelos, metamodelos e transformações têm sido propostas na literatura [18]. Além disso, para mitigar a complexidade desses ambientes, várias ferramentas que auxiliam nas tarefas de Engenharia de Software (ES) foram propostas [12]. Essas ferramentas ajudam os desenvolvedores a aproveitar experiências anteriores ao especificar um novo software, enquanto ferramentas com base em modelos ajudam a capturar, organizar e armazenar a maioria das informações trocadas como ativos de software reutilizáveis.

Para ajudar na integração de ferramentas usadas na produção de software, muitas abordagens foram propostas interoperando dados entre serviços [7]. As abordagens foco destes trabalhos caracterizam-se por minimizar os problemas relacionados ao processo de integração. Elas são adotadas nos contextos de ES, tipicamente implementadas por meio de uma arquitetura comum para serviços [34], o que demanda um protocolo de comunicação comum entre diferentes ferramentas.

Nesse contexto, uma emergente especificação industrial caracterizada como arquitetura comum para ferramentas de Engenharia de Software foi proposta: *Open Services for Lifecycle Collaboration (OSLC)* [26]. O OSLC é um padrão aberto para interoperabilidade de ferramentas de software, o qual define um modelo de representação comum para os artefatos produzidos por meio dos domínios do projeto, bem como métodos que permitem ferramentas compartilhem dados entre si.

Nesse sentido, diversas abordagens têm sido propostas para a *Continuous Software Engineering (CSE)*, em especial aquelas que integram ferramentas com o padrão OSLC. Estabelecer cadeias de ferramentas OSLC exige dos engenheiros de software o conhecimento de como criar interfaces de ferramentas para a troca de dados interoperada por meio de adaptadores. Essa tarefa não é fácil e requer etapas bem definidas que ajudam os engenheiros de software na automação do CSE.

Com base nisso, este trabalho apresenta uma pesquisa em andamento com caráter exploratório de uma ferramenta com base em *Model Driven-Development (MDD)* para integração de ferramentas

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

com o OSLC. A abordagem MDD permite a geração de código-fonte de soluções de integração, o que pode ajudar na assimilação da teoria da integração e prática. Além disso, o uso de *Domain-Specific Language (DSL)* aumentam a produtividade e melhoram a comunicação entre os especialistas do domínio.

Este trabalho é organizado como segue: A Seção 2 apresenta a metodologia e caracteriza cenário do nosso estudo. A Seção 3 aborda tecnologias investigadas durante a realização deste trabalho. A Seção 4 descreve o processo do trabalho executado em nosso estudo. A Seção 5 caracteriza o estudo conduzido no Eclipse Lyo e a Seção 6 discute os trabalhos relacionados sobre o uso Lyo na indústria. A Seção 7 discute nossa experiência de trabalho e discute tópicos de pesquisa em OSLC. A Seção 8 apresenta as limitações deste trabalho e possíveis alternativas para sua execução. Por fim, a Seção 9 apresenta as ameaças à validade do protocolo que motivou esta pesquisa e a Seção 10 conclui este trabalho.

ID	Ferramenta	Contexto	URL
T01	Redmine	gerenciamento de projeto	http://www.redmine.org
T02	Libreoffice /Excel	Gerenciamento de requisitos	http://pt-br.libreoffice.org
T03	Testlink	Gerenciamento de testes	http://testlink.org
T04	Mantis	Gerenciamento de testes	https://www.mantisbt.org
T05	GLPI	Gerenciamento de serviços de TI	https://glpi-project.org/pt-br
T06	Subversion	Controle de versões	https://subversion.apache.org
T07	Eclipse	Ambiente de desenvolvimento	https://www.eclipse.org

Table 1: Ferramentas levantadas para integração no contexto do DTIC - Unipampa

2 METODOLOGIA DE ESTUDO

Esse relato de experiência considera apenas um tipo de estudo empírico: estudo de caso. Runeson *et al.* [27] afirmam que os estudos de caso abrangem estudos bem organizados na área até pequenos exemplos, mas o consenso é que de ocorram no mundo real e conduzidos com baixo controle em comparação com estudos de experimentos controlados [32]. O termo estudo de caso também é usado para descrever um estudo de campo e um estudo observacional. Portanto, diferentes taxonomias devem ser adotadas para a caracterização deste tipo de estudo, tais como: (i) Estudo exploratório, o qual busca descobrir o que está acontecendo, buscar novos *insights* e gerar ideias e hipóteses para novas pesquisas; (ii) Descritivo, retratando uma situação ou fenômeno; (iii) Explanatório, o qual busca uma explicação para uma situação ou problema, principalmente, mas não necessariamente, na forma de uma relação causal; (iv) e Melhoria, tentando otimizar um determinado aspecto do fenômeno estudado.

Nosso estudo foi construído com base em uma metodologia para estudos de caso do tipo Exploratório, executado do ponto de vista do pesquisador. No entanto, por motivos de espaço, não o relatamos como um estudo de caso, mas sim como um relato de experiência.

Nosso objetivo é analisar na prática uma parte de um cenário dessa organização, derivando um processo para desenvolvimento de integradores e identificando as dificuldades que uma equipe em cenário real enfrentará ao utilizar o aparato ferramental explorado. Trata-se de um estudo fundamental que relata a nossa experiência ao longo de um ano e meio na investigação do tema, dedicado tanto para alinhamento conceitual, planejamento e execução de uma integração que faz uso de dados de um projeto de software real.

Portanto, é um relato de experiência que embasará os demais tipos de estudos de caso planejados para aplicação em ambiente real de desenvolvimento, incluindo estudos derivados como o Descritivo, o Explanatório e o de Melhoria. Ao melhor do nosso conhecimento, é o primeiro relato que busca caracterizar os elementos essenciais para a integração de ferramentas em vistas à uma futura implementação de engenharia de software contínua por uma fábrica de software governamental.

2.1 Caracterização do Cenário

As ferramentas que compõem a cadeia de ferramenta à serem exploradas neste estudo são mostradas na Tabela 1. Elas foram selecionadas com base no ecossistema de software da Diretoria de Tecnologia da Informação e Comunicação (DTIC), da Universidade Federal do Pampa (Unipampa). Esse setor de desenvolvimento da Unipampa trabalha com softwares livres e de código aberto e possui ferramentas que suportam atividades de desenvolvimento, gestão e comunicação. Dentre o aparato ferramental utilizado, este estudo teve por objetivo testar OSLC e Eclipse Lyo em um conjunto de três ferramentas que melhor representem um cenário de Engenharia de Software Contínua para o DTIC.

2.2 Foco de Contribuição

A implementação de um adaptador OSLC requer que muitas decisões sejam tomadas pelos desenvolvedores. Por exemplo, deve-se identificar quais os dados devem ser providos/consumidos, quais especificações de domínios utilizar, qual o fluxo da troca de mensagens, dentre outros [20]. Nesse sentido, com o objetivo de explorar a ferramenta Eclipse Lyo, buscou-se desenvolver uma solução de integração OSLC com base na topologia ponto a ponto por meio de adaptadores OSLC. A topologia de integração ponto a ponto estabelece uma conexão direta entre duas ferramentas [17]. Além disso, essa topologia é caracterizada por aplicações fortemente acopladas, portanto, quando uma ferramenta é substituída, é necessário refazer o processo de integração, resultando em significativo custo operacional.

A fim de explorar e ilustrar os benefícios da geração de códigos de adaptadores para ferramentas de diferentes domínios, foram selecionadas três ferramentas utilizadas pelos profissionais de desenvolvimento, sendo elas: Libreoffice/Excel, Redmine e Testlink.

2.3 Protocolo de Pesquisa

Este estudo, de caráter exploratório, possui o objetivo de identificar os aspectos envolvidos na modelagem de integração através do Eclipse Lyo e geração e códigos com base em artefatos em conformidade com o padrão OSLC. Para isso, pretende-se argumentar sobre

elementos necessários ao longo do processo de aprendizado para se usar a interface do ambiente Eclipse, identificando as atividades necessárias para implementar adaptadores e mapear as vantagens e desvantagens do uso do Eclipse Lyo. Para tanto, elaborou-se um protocolo de pesquisa[1], o qual serviu como guia para a execução do estudo e relato da experiência. Este protocolo apresenta questões de pesquisa bem definidas, as quais foram suprimidas neste relato por motivos de espaço.

3 CONCEITOS E EMBASAMENTO TEÓRICO

3.1 Integração de Ferramentas

A área de estudo sobre integração de ferramentas possuem contribuições desde a década de 1980. Wasserman [30] propõe em seu estudo cinco níveis de integração de ferramentas: (i) plataforma, (ii) controle, (iii) processo, (iv) dados e (v) apresentação. Relacionado a integração de dados, existem desafios para a troca de dados como manter a consistência e a rastreabilidade dos dados.

Integração de ferramentas em ambientes de Engenharia de Software trata sobre como ferramentas com características heterogêneas podem ser compatíveis em diferentes aspectos. Esses aspectos incluem os formatos dos dados, convenções de interface do usuário, funcionalidades da aplicação e outros aspectos do desenvolvimento da ferramenta [29].

Ainda que o maior desafio seja prover um ambiente integrado de forma automatizada que compreenda todo o ciclo de vida do software [11], existem diversas abordagens para integrar ferramentas de software. Cada uma dessas abordagens envolve uma série de características que devem ser levadas em consideração durante sua escolha [17].

3.2 Open Services for Lifecycle Collaboration

Open Services for Lifecycle Collaboration é um padrão aberto para integração de ferramentas de software. O OSLC define uma forma de representação comum para artefatos, bem como métodos para o compartilhamento de dados em todos os domínios do projeto. As especificações OSLC consistem em regras para criar, atualizar, recuperar e ligar ativos de software estruturados nos formatos RDF/XML e JSON. Existem também os domínios OSLC que estendem a especificação principal e definem como representar artefatos em domínios como gerenciamento de requisitos, gerenciamento de qualidade, gerenciamento de configuração e mudanças, etc.

O OSLC têm como base os princípios dos dados ligados e segue as regras definidas por Tim Berners-Lee [21] para ligar dados na web. Nesse sentido, os relacionamentos entre os artefatos em uma cadeia de ferramentas podem ser estabelecidos sem a duplicação de dados por meio de *links*. Esses *links* são mantidos em uma estrutura chamada *Triple* que consiste em dados de sujeito-predicado-objeto que descrevem o relacionamento entre artefatos, partes interessadas e atividades. Por exemplo, em um ambiente formado pelas ferramentas de gerenciamento de requisitos (Ferramenta A) e gerenciamento de testes (Ferramenta B), um requisito na Ferramenta A pode ser validado por um caso de teste na Ferramenta B por meio do OSLC.

Em uma cadeia de ferramentas OSLC, uma aplicação pode ser classificada como Provedora ou Consumidora. Provedores destinam-se a armazenar e fornecer dados, permitindo aos consumidores acesso fácil para navegar, criar e recuperar dados [5].

Existem três abordagens para prover suporte OSLC em ferramentas de software: abordagem nativa, plugin e adaptador. A abordagem de suporte nativo é recomendada para desenvolvedores de ferramentas. As abordagens plugin e adaptador são semelhantes, entretanto a construção de plugins são recomendadas apenas nos casos em que há o conhecimento das tecnologias que envolvem o desenvolvimento da ferramenta, como linguagem de programação e arquitetura. Em outros cenários, a abordagem recomendada é a construção de adaptadores OSLC. Nesse contexto, surge a necessidade de abordagens para se implementarem adaptadores OSLC, como por exemplo MDD.

3.3 Eclipse Lyo

O Eclipse Lyo é um projeto aberto que visa ajudar a comunidade interessada em integrações a adotar as especificações OSLC em suas ferramentas [13]. O Lyo possibilita o desenvolvimento de novas soluções compatíveis com o OSLC por meio da abordagem de desenvolvimento dirigido por modelos, a qual permite que engenheiros possam trabalhar com um nível maior de abstração por meio de DSLs.

O projeto consiste nos seguintes componentes: OSLC4J SDK, que é um kit de desenvolvimento Java para interfaces de ferramentas provedoras e consumidoras OSLC e o Lyo Designer, que é um gerador de códigos baseado em modelos, permitindo a representação gráfica dos modelos de integrações. Entretanto, o código gerado possui apenas um conjunto de métodos que permitem a comunicação entre as ferramentas e para acessar os dados armazenados internamente é necessário implementá-los manualmente.

Existem três perspectivas para a representação de modelos no Lyo Designer: Perspectiva Especificação de Domínios, Perspectiva Cadeia de Ferramentas e Perspectiva Interface do Adaptador. Cada perspectiva permite representar um tipo de diagrama, como visto a seguir:

(1) **Perspectiva Especificação de Domínios -** Trata sobre a representação dos domínios OSLC. É possível adicionar múltiplos domínios. Cada domínio é composto por *Resources* (artefatos) e *Resources Properties* (valores permitidos, cardinalidade e opcionalidade).
(2) **Perspectiva Cadeia de Ferramentas -** Refere-se a estrutura da cadeia de ferramentas e artefatos que serão compartilhados entre as aplicações, definindo quais as ferramentas que serão consumidoras e provedoras.
(3) **Perspectiva Interface do Adaptador -** Representa a estrutura dos adaptadores de cada ferramenta representada na perspectiva Cadeia de Ferramentas. Essa estrutura segue a especificação principal do OSLC, sendo a base para a geração de códigos.

O processo para desenvolver um adaptador OSLC no Eclipse Lyo consiste nas seguintes etapas: (i) Criar no ambiente Eclipse um projeto *Modelling Project*; (ii) Representar graficamente os modelos da cadeia de ferramentas OSLC; (iii) Validar se a representação dos modelos está de acordo com a regras do metamodelo; (iv) Gerar

[1] https://drive.google.com/drive/folders/1iX0fs4Od7Kr53bzsJjM3nHtQmyE5O16h?usp=sharing

os códigos de forma automática; (v) Implementar manualmente o código necessário para acessar dados de outra ferramenta; (vi) Adicionar o código gerado em um projeto Maven Web no ambiente Eclipse; (vii) Executar o adaptador.

Após realizar as três perspectivas de modelagem, as abordagens baseadas em MDD permitem gerar código para adaptadores OSLC, promovendo assim a integração automatizada de um modelo independente para uma representação específica da plataforma.

4 METODOLOGIA PARA DESENVOLVIMENTO DA INTEGRAÇÃO

Este tipo de estudo pode e deve ser replicado em ambientes de desenvolvimento reais. Para tal, é essencial a definição de uma metodologia que sintetiza toda a bagagem adquirida ao longo da execução da integração em vistas à implementação da Engenharia de Software Contínua. Por um motivo de fomentar a transferência de conhecimento tácito em explícito, derivou-se um conjunto de atividades à ser desempenhada pelos engenheiros de software.

Assim, estabeleceu-se um processo para o desenvolvimento de adaptadores OSLC no Eclipse Lyo, ilustrado na Figura 1. Para seu início, é necessário que os desenvolvedores conheçam o escopo da cadeia de ferramentas, definindo quais vão compor a solução de integração. Ou seja, parte-se do ponto onde o conjunto de ferramentas mostrados na Tabela 1 está bem definido pela organização/fábrica de software, também de que ela traçou seu objetivos em termos de automação dos processos de desenvolvimento e de que seus times utilizam estas ferramentas em seu quotidiano. Portanto, o foco do processo é exclusivamente destinado para o perfil integrador, o qual viabilizará a colaboração em *toolchain* por meio de adaptadores que operem conforme o padrão OSLC.

Com esse cenário definido, o próximo passo é estabelecer os relacionamentos entre as ferramentas. Nesse subprocesso/atividade, são definidos o fluxo da troca de dados ao longo da cadeia de ferramentas. A integração de ferramentas com OSLC é estabelecida por meio da topologia ponto a ponto. Dessa forma, cada uma das ferramentas possuem uma ou mais ligações com as demais. Essas conexões representam o fluxo da toca de dados. Nesse sentido, em uma solução OSLC, as ferramentas podem ser classificadas como ferramentas provedoras ou consumidoras. Cada uma das ferramentas podem pertencer a um ou outra, ou então ambas categorias. Relacionado a isso, é necessário identificar quais os artefatos serão compartilhados entre as ferramentas. Por exemplo, uma ferramenta de gerenciamento de requisitos pode consumir dados de uma ferramenta de testes, enquanto uma ferramenta de análise estática de código-fonte pode prover relatórios e consumir requisitos ao mesmo tempo. Assim, não é obrigatório que todos os artefatos de uma ferramenta sejam compartilhados via OSLC.

Identificando os artefatos que serão compartilhados, ainda resta selecionar quais os atributos serão representados e compartilhados via OSLC. Essa atividade geralmente é realizada com base na interface de usuário da ferramenta. Isso não significa que são obrigatórios que todos os atributos sejam representados em OSLC. Por isso, o filtro das propriedades essenciais é importante para o restante do processo. Por exemplo, em um cenário talvez não seja interessante representar a prioridade de um caso de teste.

Além dos relacionamento em nível de ferramenta, é necessário estabelecer quais os artefatos devem ser relacionados através dos dados ligados. Este relacionamento de propriedades no formato RDF/XML em que o artefato é transformado para ser compartilhado.

O consórcio de empresas e acadêmicos que mantém o OSLC propuseram uma série de domínios OSLC. Esses domínios significam que na teoria, quaisquer ferramenta que pertença a uma fase do ciclo de vida do software (*e.g.* requisitos, testes), mantém artefatos que podem ser representados por essas especificações. Essas informações não estão disponíveis no Eclipse Lyo, sendo necessário acessar as várias páginas webs correspondentes aos domínios. Com base nisso, para escolher os domínios OSLC devem estar de acordo com os domínios das ferramentas selecionadas.

Cada um dos domínios possuem uma série de *resources* que são representações dos artefatos. Por exemplo, o domínio de Gerenciamento de Testes possui os *resources* de Caso de Testes, Plano de Testes, etc. Além disso, esses *resources* possuem propriedades/atributos correspondentes aos artefatos.

Além dos domínios principais, existem os domínios auxiliares que servem para representar outros tipos de dados. Por exemplo, os domínios FOAF e dcterms são padrões para representar dados pessoais na web. Por exemplo, para representar o criador de um plano de testes, o domínio de gerenciamento de qualidade não possui um *resources* pessoa para tal, sendo necessário referência esses domínios auxiliares.

Outra atividade essencial é definir a estrutura dos adaptadores OSLC. Para isso, é necessário seguir a especificação principal (Core) do OSLC. Nessa especificação, todos os artefatos são compartilhados via serviços. Para isso, cada um deles possui um *Uniform Resource Identifier (URI)*. Com isso, por meio de operações Restful é possível acessá-los e manipulá-los através de funções como *Get* e *Post*.

Com base nesse cenário, a fim de organizar o projeto e permitir o acesso a esses serviços, deve-se definir a estrutura desses adaptadores através de catálogos. Esses catálogos disponibilizam URIs para os provedores de serviços e seus serviços. Os provedores de serviços permitem organizar os serviços de forma que seja atrativo aos interesses do projeto. Por exemplo, é possível filtrar os defeitos de uma ferramenta *bugtracker* através dos projetos. Ou ainda, filtrar as tarefas em uma ferramenta de gestão de projetos pelos times.

Com essas atividades planejadas e efetuadas, é possível gerar o código dos adaptadores no Lyo. Entretanto, para que seja possível manipular os dados, primeiro deve-se buscar e estudar uma API RESTful para cada uma das ferramentas envolvidas. Caso não seja possível localizar, deve ser necessário criar sua própria API para a extração dos dados.

Ao final, após validar o projeto e gerar os códigos, percebe-se que apenas os esqueletos dos código foram gerados, sendo necessário implementar a lógica de cada um deles. Para isso, são implementadas as funções REST com base na API selecionada para a ferramenta.

5 DEMONSTRAÇÃO CONCEITUAL

Esta seção detalha a implementação de adaptadores OSLC por meio da geração de código-fonte com base em modelos no Eclipse Lyo. A Figura 2 mostra a cadeia de ferramentas do nosso estudo exploratório, composta por três ferramentas de diferentes domínios,

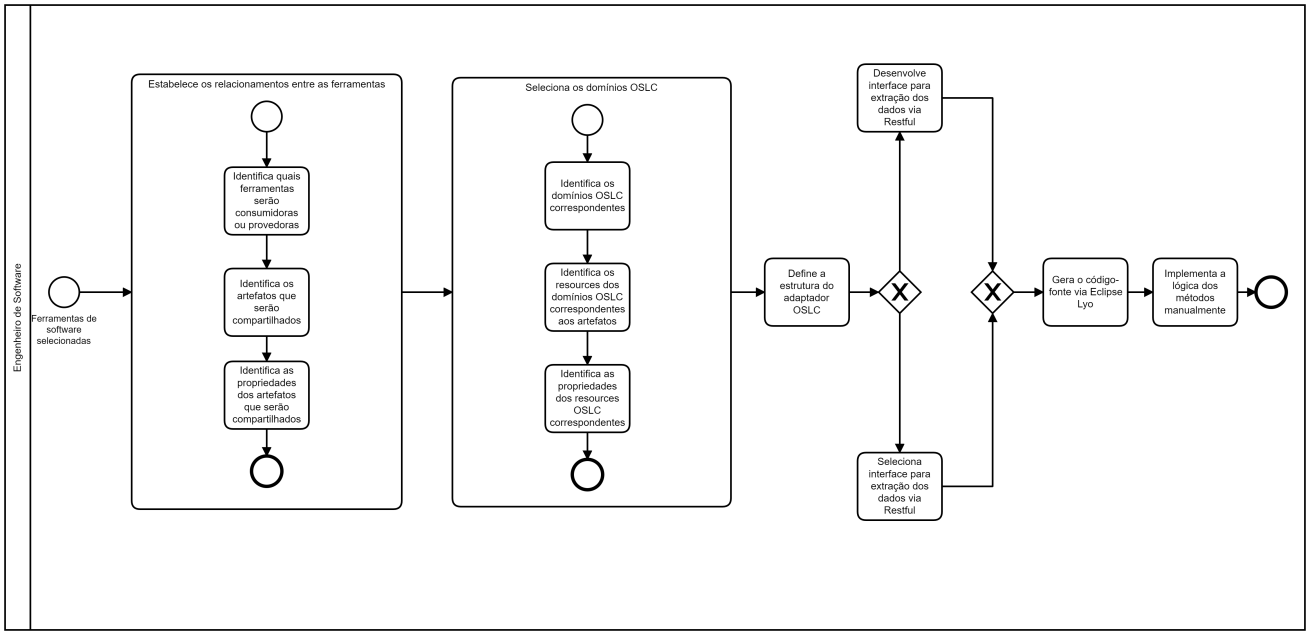


Figure 1: Processo para o desenvolvimento de elementos para a integração de ferramentas em OSLC e Eclipse Lyo

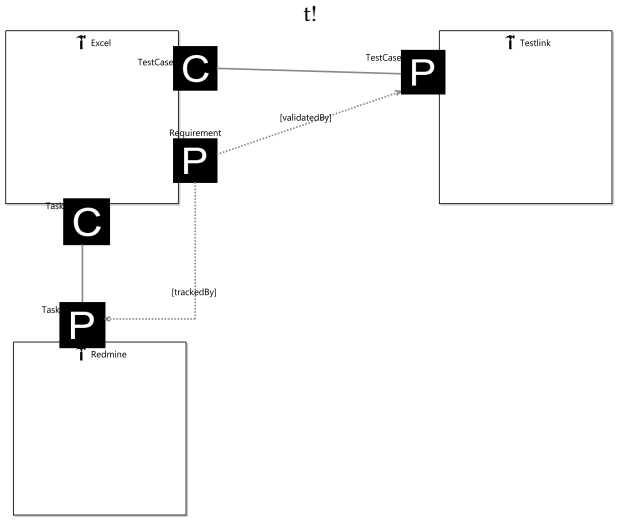


Figure 2: Perspectiva Cadeia de Ferramentas

sendo elas Excel para o gerenciamento de requisitos, Redmine para o gerenciamento de projetos e Testlink para gerenciamento de testes.

A ferramenta Excel mantém requisitos de software e deseja-se ligar através das propriedades dos dados ligados do OSLC às tarefas gerenciadas pelo Redmine e aos casos de testes do Testlink. Com base nisso, o adaptador OSLC do Excel possui duas interfaces consumidoras para tarefas e casos de testes respectivamente e uma interface que provê requisitos de software. Além disso, as ferramentas Redmine e Testlink possuem interfaces provedoras para o compartilhamento de dados.

Embora nosso cenário tenha definido a estrutura da cadeia de ferramentas do nosso estudo, o primeiro passo para gerar os códigos-fonte dos adaptadores é a representação gráfica dos domínios OSLC. Como mostra a Figura 3, exploramos três domínios OSLC: Requirements Management (RM) [2], Change Management (CM) [3] e Quality Management (QM) [4].

Cada um desses domínios possui propriedades e *resources* para a representação dos artefatos mantidos pelas ferramentas e seus relacionamentos. Por exemplo, o domínio RM possui o *resource* Requirement com as propriedades que o descrevem como identificador, data de criação e modificação. Além disso, nessa representação são estabelecidos os relacionamentos entre os artefatos a partir de propriedades como a `validateBy` em que é possível ligar um caso de teste que valida determinado requisito. Além dos domínios OSLC, também exploramos outros domínios auxiliares, os quais têm objetivo de padronizar informações da web, como o FOAF [5] e Dublin Core [6].

A partir de serviços OSLC, é possível recuperar, editar, atualizar e excluir artefatos de software gerados pela cadeia de ferramentas. Essas funcionalidades são modeladas na perspectiva do adaptador. A Figura 4 mostra a interface do adaptador da ferramenta Redmine. A

[2]http://docs.oasis-open.org/oslc-domains/oslc-rm/v2.1/oslc-rm-v2.1-part2-requirements-management-vocab.html
[3]http://docs.oasis-open.org/oslc-domains/cm/v3.0/cs02/part2-change-mgt-vocab/cm-v3.0-cs02-part2-change-mgt-vocab.html
[4]https://docs.oasis-open-projects.org/oslc-op/qm/v2.1/psd02/quality-management-vocab.html
[5]http://xmlns.com/foaf/spec/
[6]https://www.dublincore.org/specifications/dublin-core/dcmi-terms/

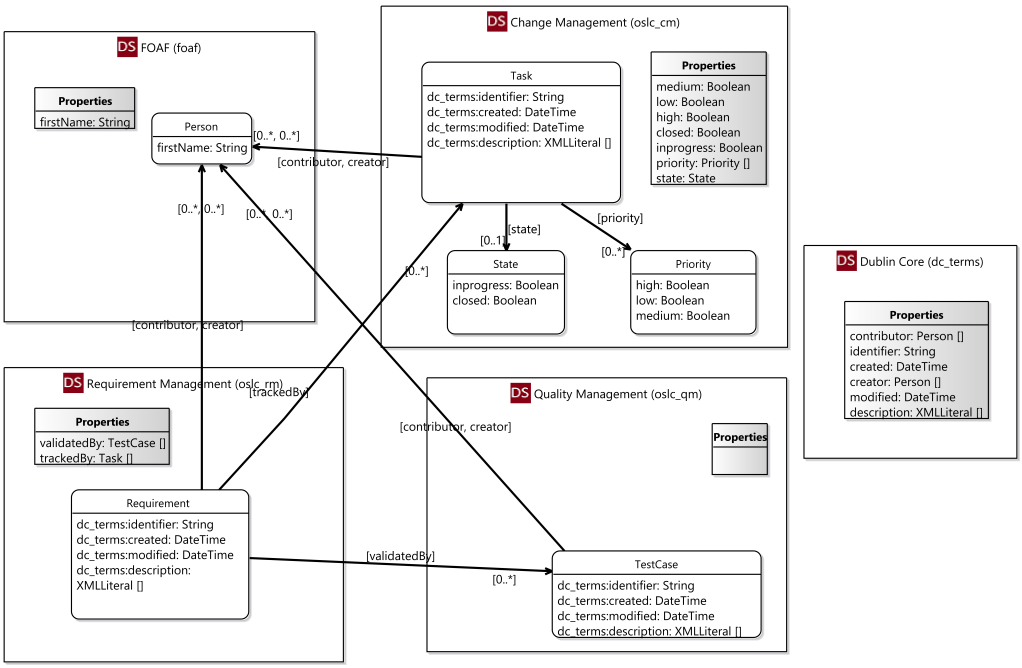


Figure 3: Perspectiva Domínios OSLC

estrutura desse adaptador segue as especificações da Especificação principal do OSLC. Em adaptadores OSLC, todos os serviços são acessados através de *Uniformd Resource Identifier* (URIs). Ao acessar o adaptador é disponibilizado um catálogo de serviços (*Service Provider Catalog*) que mantém URIs para todos os provedores de serviços (*Service Providers*), que representam os projetos cadastrados no Redmine. Cada provedor de serviço disponibiliza URIs para acessar o serviços (*Services*), que nesse cenário são as tarefas de cada um dos projetos. Ao final do fluxo, é possível acessar cada uma das tarefas e manipulá-las através das funções RESTful que foram modeladas na perspectiva da interface do adaptador.

O Lyo gera apenas os esqueletos de classes Java, sendo necessário implementar manualmente a lógica dos adaptadores. Para que isso seja possível, as ferramentas devem possuir alguma interface que possibilite manipular seus dados, como uma API RESTful. Em nosso estudo, usamos três APIs RESTful, uma para cada ferramenta envolvida no processo. Graças ao componente OSLC4J do Lyo, todos os objetos tornam-se possível de serem representados nos formatos RDF/XML e JSON, possibilitando a troca de dados entre as ferramentas após a transformação dos seus dados em objetos Java para o padrão OSLC.

A Figura 5 mostra a interface gráfica do adaptador para a ferramenta Redmine. É possível ver o URI do provedor de catálogo para todos os projetos cadastros na ferramenta. Ao acessar qualquer um dos projetos as partes interessadas terão acesso as tarefas cadastradas naquele projeto com seus respectivos dados. As tarefas são representadas em arquivos RDF seguindo a estrutura da especificação OSLC, como mostra a Figura 6. Essa representação permite

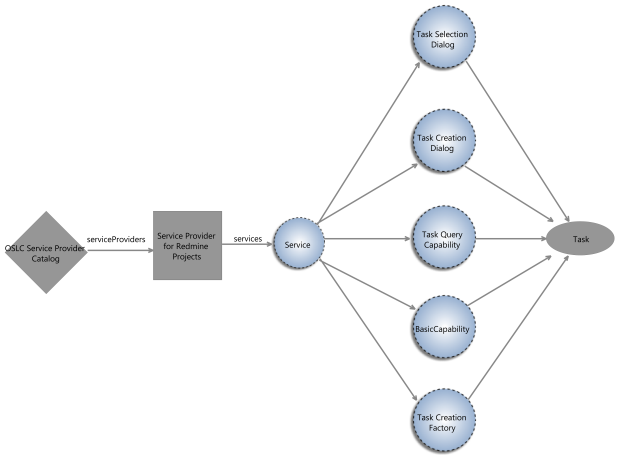


Figure 4: Perspectiva Interface do Adaptador

a troca de dados e assim, todas as ferramentas que possuem suporte ao OSLC, poderão consumi-las futuramente.

6 TRABALHOS RELACIONADOS

A principal abordagem para o desenvolvimento de adaptadores OSLC encontrada na literatura é a geração de códigos-fonte com base em modelos. Esses trabalhos reportam o uso do Eclipse Lyo na indústria como alternativa para desenvolver interfaces OSLC em soluções de integração. Nesse sentido, Marko *et al.* [22] reporta o

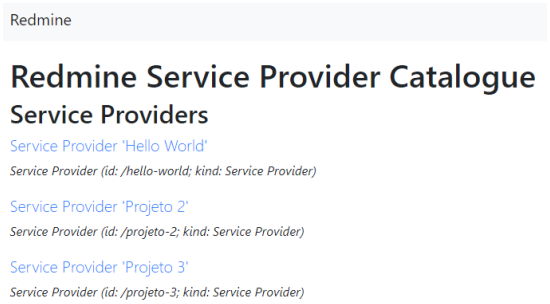


Figure 5: Interface Gráfica do adaptador OSLC para a ferramenta Redmine

desenvolvimento de um adaptador OSLC para o módulo de gerenciamento de requisitos em uma ferramenta ALM. Nardone *et al.* [25] descreve o desenvolvimento de um ambiente para testes funcionais em nível de sistema para controladores ferroviários integrado com adaptadores OSLC.

Existem trabalhos que buscam integrar ambientes com base no contexto de sistemas críticos de segurança [10] [14] [16] [34]. Esses ambientes são conhecidos por possuírem atividades de verificação e validação, além de simulações e do grande volume de dados.

Outros trabalhos focam na avaliação do Eclipse Lyo como solução para a geração semi-automática de código de interfaces OSLC através da representação de metamodelos EMF [8] [15]. Esses trabalhos discutem o desenvolvimento de adaptadores para ferramentas de análise e projeto, motivados pela heterogeneidade dos modelos gerados por essas ferramentas. Apesar disso, apenas esqueletos de código são gerados e precisam ser finalizados manualmente. Essas abordagens já foram validadas em contextos reais e têm vários pontos positivos sobre a escolha de abordagens com base em modelos como o reuso dos modelos dos adaptadores em outras ferramentas e a de não ser necessário conhecer a tecnologia interna dessas ferramentas.

Além do Eclipse Lyo, encontramos o trabalho de Biehl *et al.* [9] que propõe o desenvolvimento de adaptadores OSLC para integrar ferramentas web e *desktop*, utilizando a DSL *Tool Integration Language (TIL)*. Esse trabalho é motivado pela atividade de substituição de ferramentas *desktop* por soluções web para que essas cadeias de ferramentas se mantenham estáveis nesse período de transição.

Relacionado os processo que executamos para a construção dos adaptadores, Baumgart *et. al* [6] propõe uma série de passos para integrar ferramentas através de uma especificação de interoperabilidade (IOS). Os autores apresentam atividades que envolvem selecionar os resources e atributos OSLC para ferramentas baseadas em metamodelos, como o Simulink e Papyrus. O processo tem início com um conjunto de ferramentas bem definido e a partir disso, as atividades tem como objetivo identificar os domínios e propriedades OSLC correspondentes aos artefatos das ferramentas. A implementação dos adaptadores também é realizada no Eclipse Lyo.

Não existe uma receita para se desenvolver adaptadores OSLC. Entre os aspectos em comum encontrados nos trabalhos relacionados estão que os autores seguem as especificações OSLC rigidamente, buscando mapear os atributos dos artefatos compartilhados com os domínios OSLC e apesar de ser uma alternativa viável para a construção de adaptadores, seu uso na indústria indica não ser uma atividade trivial, necessitando de desenvolvedores especialistas no domínio.

Embora o padrão OSLC seja recomendado para a integração de ambientes que adotam abordagens como DevOps e ALM, o OSLC não é explorado em soluções que cobrem todo o ciclo de vida do software, limitando-se apenas em algumas fases do desenvolvimento. Além disso, as soluções propostas pelos trabalhos encontrados na literatura indicam que há o interesse da indústria para o aprimoramento da geração de código de adaptadores OSLC por meio da representação de modelos.

7 RELATO DE EXPERIÊNCIA E DISCUSSÕES

Nosso objetivo principal foi o de identificar o suporte existente para integração de ferramentas de Engenharia de Software, com atenção especial para integração de ferramentas relacionadas com cenários de teste. Ou seja, executou-se um estudo prático abrangendo num escopo maior do que o das ferramentas configuradas em *pipelines* de IC. Portanto, uma ferramenta de IC é exemplificada como uma das partes componentes do cenário motivado, não como o fim.

Embora algumas abordagens para estabelecer a rastreabilidade entre artefatos sejam encontradas na literatura [24] e para o controle de versões [4][19] utilizando o OSLC, essas abordagens baseiam-se no nível de integração de artefatos em uma cadeia de ferramentas, limitando-se apenas a consulta desses dados.

Ferramentas de IC permitem a automação de atividades em um repositório central que permite que apenas códigos testados sejam integrados ao produto final. Entretanto, essas ferramentas precisam ser configuradas para se adequarem ao ambiente, o que envolve engenheiros de software especialistas do domínio. Ainda, cada ferramenta de IC possui propriedades e atividades próprias para sua execução. Essas configurações definem o fluxo das atividades e o relacionamento entre os artefatos/ferramentas envolvidas na implantação de servidores de IC. Por exemplo, a ferramenta CircleCI segue o fluxo de configurações como build, test1, hold e deploy. Enquanto isso, na ferramenta os fluxo define-se como: 1) Job; 2) Build; 3) Artifact; 4) Workspace; 5) Status e 6) Trend.

Com base nisso, o OSLC possui o potencial para representação e padronização dessas configurações. Além disso, o consórcio que mantém o OSLC propôs o domínio de automação, que atualmente está em sua versão 2.0, para atividades que envolvem planos de automação, automação de requisições e resultados de automação do desenvolvimento de software, teste, implantação e ciclo de vida de operações. Nesse sentido, o fluxo de configurações pode ser representadas através do domínios OSLC ou na extensão deles. Nesse sentido, uma DSLs com base em OSLC poderia suprir a necessidade da geração automática dessas configurações por meio de modelos. Ainda, a literatura sugere que o uma das motivações para a adoção do OSLC seja sua capacidade de estabelecer relacionamentos entre artefatos de software, inclusive de domínios diferentes do projeto.

Outra possível aplicação do OSLC em IC está no próprio fluxo de atividades de automação. Domínios OSLC como o Gerenciamento de Qualidade possuem propriedades que descrevem os resultados dos diversos níveis de testes. A partir dessas informações, é possível automatizar a atividade de implantação, permitindo que apenas os

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oslc_data="http://open-services.net/ns/servicemanagement/1.0/"
    xmlns:oslc="http://open-services.net/ns/core#"
    xmlns:j.0="http://open-services.net/ns/cm#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:dcterms="http://brunomarcelo/dc/terms/"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:oslc_cm="http://open-services.net/ns/cm#Task">
    <j.0:TaskTask rdf:about="http://localhost:8080/redmine-adaptor/services/serviceProviders/hello-world/tasks/11">
        <dcterms:identifier>11</dcterms:identifier>
        <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
    >2020-06-23T05:43:14Z</dcterms:created>
    </j.0:TaskTask>
</rdf:RDF>
```

Figure 6: Representação das Tarefas na especificação OSLC

Table 2: Propriedades de Integração dos Trabalhos Relacionados

Estudo	Ano	Ferramentas Integradas	Fases de Desenvolvimento Exploradas	Artefatos Compartilhados
Biehl et al. [9]	2013	Bright Green Projects, Matlab Simulink	Requisitos e Análise & Projeto	Requisitos e Modelos
Zhang and Møller-Pedersen [34]	2014	Ferramenta para modelos Simulink, UML e especificação IEC 61131 e	Análise e Projeto	Modelos
Biehl et al. [8]	2014	Matlab Simulink	Análise e Projeto	Modelos
Baumgart and Ellen [6]	2014	IBM Rational Doors, Papyrus e Matlab Simulink	Requisitos, Análise & Projeto, Testes	Requisitos e Modelos
Marko et al. [22]	2015	HP Quality Center, ReFine e VeVat	Requisitos e Testes	Requisitos escritos em linguagem natural e semi-formal
Biró et al. [10]	2017	RTCT (Requiremets Traceability and Consistency cheking Tool)	Requisitos	Requisitos
Gürdür et al. [16]	2018	Sistemas ciber-físicos	Análise e Projeto	Modelos
El-khoury et al. [14]	2019	Sesammtool, D2RQ	Requisitos,	Análise & Projeto Requisitos e Modelos
Nardone et al. [25]	2020	RailModel GUI, IBM Rational Doors, IBM Rational Quality Manager, IOP Test Writer	Requisitos, Testes	Requisitos, Casos de Testes, Planos de Testes, Scripts de Teste, Registros de Execução de Testes, Resultados de Testes
Nosso estudo	2020	Libreoffice/Excel, Redmine e Testlink	Processo do Projeto, Requisitos e Testes	Requisitos Funcionais, Requisitos não Funcionais, Casos de Uso, Tarefas e cronograma, Planos de Teste e Casos de Teste

códigos em que foram aprovados nas atividades de testes usando o OSLC como gatilho para a execução das atividades.

Esse cenário ilustra a necessidade de uma representação comum para artefatos de software se deseja-se automatizar o processo de software, inclusive em atividades de configuração desses ambientes. Uma vez que a Engenharia de Software Contínua necessita de um escopo maior do que a IC, buscou-se explorar este cenário por meio da geração automática de código devotado para a integração de ferramentas. Assim, nosso estudo foca no relato de experiência na análise prática de apenas uma abordagem para projeto e desenvolvimento de cadeia de ferramentas: o Eclipse Lyo.

8 LIMITAÇÕES

A Tabela 2 demonstra que todos os trabalhos que vem explorando o uso de OSLC na integração de aplicações para a Engenharia de Software Contínua são limitados, incluindo o nosso. Ou seja, os trabalhos existentes conseguem explorar uma parte muito pequena da integração quando se considera todo o ecossistema de software utilizado nas organizações desenvolvedoras de software. Nossa contribuição também tem um limite quando considera-se o restante das ferramentas utilizadas pelo DTIC e, como constatou-se, boa parte delas não possuem integração nativa especificada em OSLC. Portanto, é importante salientar que o estudo exploratório atende apenas uma parte do necessário pelo DTIC para uma automação completa do processo de desenvolvimento desta organização.

Nosso estudo também é limitado para apenas uma DSL que permite a representação de elementos de integração. Além do Eclipse Lyo, existem outras alternativas para implementar soluções de integração de ferramentas baseadas em OSLC consideradas que se considera para futuros estudos exploratórios, talvez até mesmo o desenvolvimento de uma DSL própria.

9 AMEAÇAS À VALIDADE

O planejamento de um estudo de caso é flexível e pode sofrer alterações para se adequar ao cenário. A seguir são apresentadas algumas possíveis ameaças à validade do trabalho, que foram estruturadas conforme [32]:

Ameaças ao Construtor: Ameaças ao construtor levam em consideração o quão bem o estudo é planejado para responder ao objetivo principal. Para ter validade, o plano de estudo considera a integração de ferramentas em OSLC e Eclipse Lyo considerando necessidades reais. Em 2019, durante uma disciplina do curso de pós-graduação em Engenharia de Software dedicou-se cinco aulas, de quatro horas cada uma, para introduzir os alunos no contexto de Engenharia de Software Contínua por meio de ferramentas que fomentam a automação. Em especial, uma aula foi dedicada para

mecanismos disponíveis para integração de ferramentas. Na ocasião, seis alunos do mestrado, que pertencem ao time do DTIC, forneceram *feedback* sobre a possibilidade de automatizar os processos, interessando-se e abrindo espaço para execução de um estudo de caso. Na ocasião, discutiu-se as ferramentas utilizada pelo DTIC, o que possibilitou realizar um planejamento inicial do protocolo. O presente estudo pretendia ser aplicado em cenário real, no entanto a pandemia fez com que o time se dissipasse, trabalhando de modo remoto. Assim, os métodos de avaliação quantitativos e qualitativos inicialmente previstos não puderam ser conduzidos, caracterizando um cenário que apresentaria fatores de confusão demasiados para se gerenciar. Nesse sentido, não foi possível resolver uma ameaça à validade: dados insuficientes para responder questões de pesquisa, os quais necessitariam de um método estatístico adequado. Identificada a impossibilidade de um estudo de caso com a interação de times, desconsiderou-se a análise estatística e optou-se por conduzir um estudo de integração, baseado num cenário real, porém independente do *feedback* do time da organização. Tipicamente, estudos de caso exploratórios não possuem questões de pesquisa bem definidas, sendo assim dependentes da avaliação de um cenário que seja significativo para extrair aprendizados. Assim, alguns autores consideram o tipo de estudo como uma ameaça à validade por si [32]. Portanto, para mitigar essa ameaça, buscou-se exercitar ferramentas do cenário da organização e caracterizar o estudo como um relato de experiência, diminuindo assim a sua força empírica.

Ameaças Internas: As ameaças internas consideram escolhas que consigam responder adequadamente as questões de pesquisa, que em nosso caso são mapeadas para um objetivo único. A escolha das ferramentas se deu por conveniência pelo relato dos profissionais da organização. No entanto, para análises mais precisas, considera-se que além das ferramentas de software livre e de código aberto que compõem o ecossistema de software desta organização, pode ser necessário integrar as ferramentas do nosso estudo com outras aplicações produzidas internamente não mapeadas, como as que discutimos em diversos cenários, incluindo: ferramentas de teste de desempenho [1], ferramentas de teste como um serviço [2], ferramentas de teste para Apps [3], dentre outras ferramentas levantadas em mapeamos e revisões sistemáticas em nosso grupo de pesquisa[7]. Uma vez que se trata de um relato de experiência, moldado conforme o cenário de pandemia, não pôde-se resolver a ameaça à validade decorrente do número de participantes, caracterizado por apenas três integrantes com a perspectiva do pesquisador, e não do time da organização.

Ameaças Externas: As ameças externas dizem respeito à quão generalizáveis são os resultados. O relato de experiência em questão é direcionado para apenas três disciplinas do Rational Unified Process (RUP). Logo, as observações não podem ser generalizadas para ferramentas encontradas em todo o ciclo de vida de aplicações de um processo de desenvolvimento de software completo. De qualquer modo, o estudo apresentado explora um cenário bem complexo em termos de Engenharia de Software Contínua. Devido à complexidade para compreensão das tecnologias e suas interfaces, concordando com [33] que afirma que as ferramentas existentes não estão preparadas para a integração, e considerando as mesmas limitações dos estudos discutidos como trabalhos relacionados, considera-se relevante o escopo do estudo para se generalizar os achados no contexto das dificuldades para integração de ferramentas de Engenharia de Software usando-se OSLC.

Ameaças à Conclusão: Esta ameaça é considerada para os casos em que o número de artefatos produzidos pode não ser o suficiente para responder algumas questões de pesquisa. Este é o nosso caso e, apesar de não termos uma questão de pesquisa relatada, elas constam no pacote experimental por meio do protocolo de pesquisa. De modo à otimizar o espaço, apresentou-se no lugar apenas o objetivo: a integração das ferramentas utilizadas por uma organização governamental. O conjunto de ferramentas selecionadas, portanto, não representa uma amostra suficiente para se concluir sobre a viabilidade para esta organização, mas sim para identificar as dificuldades que a equipe enfrentará ao fazer implementações semelhantes no conjunto ferramental maior. Além disso, este é um estudo exploratório que visa identificar *insights* para outros tipos de estudo de caso. Portanto, ainda não relata os achados como um estudo de caso, que devem ter questões de pesquisa e os artefatos alvo destas questões, algo que pretende-se relatar num estudo futuro.

10 CONSIDERAÇÕES FINAIS

Esse trabalho apresentou um estudo exploratório sobre a ferramenta Eclipse Lyo com o objetivo de implementar adaptadores OSLC. Para isso, foram utilizadas abordagens para geração de código-fonte com base em modelos. Essas abordagens propõem mitigar alguns desafios que envolvem o processo de integração de ferramentas pois permitem desenvolvedores a trabalhar com um nível maior de abstração. Apesar disso, foram encontradas poucas alternativas para o desenvolvimento de adaptadores OSLC com base em DSLs na literatura.

O Eclipse Lyo surge como objeto de estudo para se atingir ambientes integrados de ponta a ponta através do OSLC. Nosso trabalho possibilitou identificar a estrutura que um projeto OSLC segue, bem como as tecnologias envolvidas no processo do desenvolvimento dos adaptadores.

Os domínios OSLC são disponibilizados em repositórios seguindo a documentação para servir como referência, entretanto são escassos os exemplos e são dispersos em fóruns da comunidade e em alguns repositórios. Essa falta de informação dificulta o trabalho principalmente de pesquisadores iniciantes sobre a integração de ferramentas com OSLC.

Outro ponto importante é de que para modelar diagramas no Eclipse Lyo é necessário o conhecimento da especificação OSLC. Ainda, para tomar decisões sobre a estrutura dos serviços web, é necessário saber o significado de alguns termos apresentados pelo OSLC, como por exemplo provedor e consumidor para modelar a interface de cada adaptador e também sobre as descrições dos domínios OSLC para decidir quais serão utilizados no projeto. Além disso, existem algumas lacunas de informações que surgem na construção de novas soluções OSLC. Algumas vezes a especificação é um pouco vaga ou está distribuída em vários documentos.

O OSLC propõe a representação global de artefatos mantidos por quaisquer ferramentas de software ao longo dos domínios do projeto. Entretanto, nem todos os atributos são cobertos pelos domínios OSLC. Por exemplo é possível manter o relacionamento entre um requisito e um caso de teste através da propriedade `validateBy` no

[7] GrupoX =oculto por motivos de doubleblind review

domínio de Gerenciamento de Requisitos mas não é possível acessar quais tarefas estão relacionadas a um determinado caso de teste, pois não existe uma propriedade dedicada a este relacionamento no domínio de Gerenciamento de Qualidade ou Gerenciamento de Configuração e Mudanças.

Dessa forma, o desenvolvimento de adaptadores OSLC não através do Lyo não foi totalmente explorado na área de Engenharia de Software. Além disso, cada cenário de integração em um ciclo de vida de aplicação de uma empresa possui particularidades que devem ser levadas em consideração na hora de tomar as decisões do projeto. Por mais que o estudo tenha considerado representações de ferramentas bastante utilizadas no desenvolvimento de software, o cenário representado jamais pode refletir um ambiente real de uma empresa.

Por fim, observou-se que o aparato ferramental de suporte propicia a geração automática de código. Entretanto, apenas os esqueletos dos códigos são gerados dessa forma. É necessário implementar manualmente o conteúdo dos métodos para que seja estabelecida a comunicação entre as ferramentas provedoras e consumidoras, possibilitando a troca dos artefatos mantidos por elas. Isso caracteriza uma limitação do aparato ferramental investigado, uma vez que a geração de 100% do código poderia ser obtida para OSLC com base na DSL investigada. Apesar disso, a partir desse estudo foi possível identificar os componentes mínimos necessários para implementar um adaptador de ferramentas, e portanto viável para a execução de um estudo exploratório em domínios de Engenharia de Software.

REFERENCES

[1] Blind 1. [n.d.]. Blind 1, year = 2020, booktitle = Proceedings of the 35th Annual ACM Symposium on Applied Computing, pages = Blind 1, location = Brno, Czech Republic.

[2] Blind 2. 2020. Blind 2. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. Blind 2.

[3] Blind 3. 2019. Blind 3. In *Proceedings of the XVIII Brazilian Symposium on Software Quality*. Blind 3.

[4] David Adjepon-Yamoah, Alexander Romanovsky, and Alexei Iliasov. 2015. A Reactive Architecture for Cloud-based System Engineering. In *Proceedings of the 2015 International Conference on Software and System Process (ICSSP 2015)*. ACM, New York, NY, USA, 77–81. https://doi.org/10.1145/2785592.2785611

[5] B. K. Aichernig, K. Hörmaier, F. Lorber, D. Nickovic, R. Schlick, D. Simoneau, and S. Tiran. 2014. Integration of Requirements Engineering and Test-Case Generation via OSLC. In *2014 14th International Conference on Quality Software*. 117–126. https://doi.org/10.1109/QSIC.2014.13

[6] A. Baumgart and C. Ellen. 2014. A recipe for tool interoperability. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 300–308.

[7] Matthias Biehl, Jad El-Khoury, Frédéric Loiret, and Martin Törngren. 2014. On the modeling and generation of service-oriented tool chains. *Software & Systems Modeling* 13, 2 (2014), 461–480.

[8] M. Biehl, J. El-Khoury, and M. Törngren. 2012. High-Level Specification and Code Generation for Service-Oriented Tool Adapters. In *2012 12th International Conference on Computational Science and Its Applications*. 35–42. https://doi.org/10.1109/ICCSA.2012.16

[9] M. Biehl, J. D. Sosa, M. Törngren, and O. Díaz. 2013. Efficient Construction of Presentation Integration for Web-Based and Desktop Development Tools. In *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. 697–702. https://doi.org/10.1109/COMPSACW.2013.123

[10] Miklós Biró, Felix Kossak, József Klespitz, and Levente Kovács. 2017. Graceful Integration of Process Capability Improvement, Formal Modeling and Web Technology for Traceability. In *Systems, Software and Services Process Improvement*, Jakub Stolfa, Svatopluk Stolfa, Rory V. O'Connor, and Richard Messnarz (Eds.). Springer International Publishing, Cham, 381–398.

[11] Alan W. Brown and Maria H. Penedo. 1992. An Annotated Bibliography on Integration in Software Engineering Environments. *SIGSOFT Softw. Eng. Notes* 17, 3 (July 1992), 47–55. https://doi.org/10.1145/140938.140944

[12] Christof Ebert. 2013. Improving engineering efficiency with PLM/ALM. *Software & Systems Modeling* 12, 3 (01 Jul 2013), 443–449.

[13] Jad El-khoury. 2016. Lyo code generator: A model-based code generator for the development of OSLC-compliant tool interfaces. *SoftwareX* 5 (2016), 190 – 194. https://doi.org/10.1016/j.softx.2016.08.004

[14] Jad El-khoury, Andrii Berezovskyi, and Mattias Nyberg. 2019. An industrial evaluation of data access techniques for the interoperability of engineering software tools. *Journal of Industrial Information Integration* (2019). https://doi.org/10.1016/j.jii.2019.04.004

[15] Jad El-Khoury, Cecilia Ekelin, and Christian Ekholm. 2016. Supporting the Linked Data Approach to Maintain Coherence Across Rich EMF Models. In *Modelling Foundations and Applications*, Andrzej Wąsowski and Henrik Lönn (Eds.). Springer International Publishing, Cham, 36–47.

[16] Didem Gürdür, Aneta [Vulgarakis Feljan], Jad El-khoury, Swarup [Kumar Mohalik], Ramamurthy Badrinath, Anusha [Pradeep Mujumdar], and Elena Fersman. 2018. Knowledge Representation of Cyber-physical Systems for Monitoring Purpose. *Procedia CIRP* 72 (2018), 468 – 473. https://doi.org/10.1016/j.procir.2018.03.018 51st CIRP Conference on Manufacturing Systems.

[17] Gregor Hohpe. 2003. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional. https://www.xarg.org/ref/a/0321200683/

[18] Anneke Kleppe, Jos Warmer, and Wim Bast. 2003. MDA Explained: The Model Driven Architecture: Practice and Promise.

[19] L. Lednicki, G. Sapienza, M. E. Johansson, T. Seceleanu, and D. Hallmans. 2016. Integrating Version Control in a Standardized Service-Oriented Tool Chain. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 323–328. https://doi.org/10.1109/COMPSAC.2016.141

[20] Andrea Leitner, Beate Herbst, and Roland Mathijssen. 2016. Lessons Learned from Tool Integration with OSLC. In *Information and Software Technologies*, Giedre Dregvaite and Robertas Damasevicius (Eds.). Springer International Publishing, Cham, 242–254.

[21] Linked Data 2006. Linked Data Web Page. https://www.w3.org/DesignIssues/LinkedData.html Accessed at April 2020.

[22] N. Marko, A. Leitner, B. Herbst, and A. Wallner. 2015. Combining Xtext and OSLC for Integrated Model-Based Requirements Engineering. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*. 143–150. https://doi.org/10.1109/SEAA.2015.11

[23] David G. Messerschmitt. 2005. *Software Ecosystem: Understanding an Indispensable Technology and Industry (The MIT Press)*. The MIT Press. https://www.xarg.org/ref/a/0262633310/

[24] Nasser Mustafa and Yvan Labiche. 2018. Using Semantic Web to Establish Traceability Links Between Heterogeneous Artifacts. In *Software Technologies*, Enrique Cabello, Jorge Cardoso, Leszek A. Maciaszek, and Marten van Sinderen (Eds.). Springer International Publishing, Cham, 91–113.

[25] Roberto Nardone, Stefano Marrone, Ugo Gentile, Aniello Amato, Gregorio Barberio, Massimo Benerecetti, Renato [De Guglielmo], Beniamino [Di Martino], Nicola Mazzocca, Adriano Peron, Gaetano Pisani, Luigi Velardi, and Valeria Vittorini. 2020. An OSLC-based environment for system-level functional testing of ERTMS/ETCS controllers. *Journal of Systems and Software* 161 (2020), 110478. https://doi.org/10.1016/j.jss.2019.110478

[26] OSLC 2020. Open Services for Lifecycle Collaboration Primer Web Page. open-services.net/resources/tutorials/oslc-primer/ Accessed at February 2020.

[27] Per Runeson and Martin Höst. 2008. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14, 2 (2008), 131.

[28] Carey Schwaber et al. 2006. The changing face of application life-cycle management. *Forrester Research* 18 (2006).

[29] I. Thomas and B. A. Nejmeh. 1992. Definitions of tool integration for environments. *IEEE Software* 9, 2 (March 1992), 29–35. https://doi.org/10.1109/52.120599

[30] Anthony I. Wasserman. 1990. Tool integration in software engineering environments. In *Software Engineering Environments*, Fred Long (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 137–149.

[31] M.N. Wicks and R.G. Dewar. 2007. A new research agenda for tool integration. *Journal of Systems and Software* 80, 9 (2007), 1569 – 1585. https://doi.org/10.1016/j.jss.2007.03.089 Evaluation and Assessment in Software Engineering.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media, Berlin, Heidelberg, Dordrecht & New York City.

[33] Betty Zakheim. 2017. How Difficult Can It Be to Integrate Software Development Tools? The Hard Truth. *InfoQ* (January 2017). https://www.infoq.com/articles/tool-integration-hard-truth

[34] W. Zhang and B. Møller-Pedersen. 2014. Modeling of tool integration resources with OSLC support. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. 99–110.

APÊNDICE C – PROTOCOLO DE ESTUDO DE CASO SUSPENSO

O protocolo do estudo de caso do Trabalho de Conclusão de Curso I previa um estudo de caso aplicado com times do DTIC. Isso não foi possível devido à pandemia de COVID-19, já que o time passou à operar em ambiente remoto. O plano original pretendia responder algumas questões, que não são exploradas pelos trabalhos relacionados, e que podem contribuir para a evolução da área de integração de ferramentas com OSLC, como segue:

Os dados ligados são úteis para a organização e em qual sentido?

O objetivo dessa questão é verificar o impacto dos dados ligados no fluxo de trabalho da organização, identificando quais as formas da organização utilizar essas informações (documentos novos criados, etapas novas no processo, etc.). Para isso, é importante identificar atividades que não são automatizadas ferramentas não estarem integradas.

O volume dos dados gerenciados pelas ferramentas interfere no desenvolvimento dos adaptadores?

Essa questão de pesquisa tem o propósito de verificar se o volume dos dados interfere na construção da solução de integração. Como a organização mantém uma grande quantidade de dados em suas bases de dados, entende-se necessário identificar se haverá problemas com o nível de acesso dos profissionais, problemas de concorrência de acessos aos servidores OSLC, etc.

Na prática, quais informações são necessárias de se conhecer sobre o funcionamento interno das ferramentas na construção de um adaptador OSLC?

Uma solução de integração baseada em adaptadores é recomendadas para desenvolvedores e em teoria não é necessário conhecer o funcionamento interno das ferramentas. Essa questão tem o objetivo de identificar quais as características das ferramentas precisam ser consideradas na construção de um novo adaptador.

Durante a execução deste TCC, foi possível identificar que é necessário conhecer uma interface Restful para extrair os dados das ferramentas. Entretanto, não foi explorado um cenário no qual foi necessário implementar uma interface do zero. Essa questão de pesquisa é importante pois organizações desenvolvem suas próprias ferramentas, sendo necessário implementar ou selecionar uma interface Restful para usar o OSLC.

É possível reutilizar os adaptadores em outras ferramentas do mesmo domínio?

Um dos pilares do OSLC é a colaboração. Baseado nisso, essa questão de pesquisa propõe identificar se é possível reutilizar os adaptadores OSLC e disponibilizá-los como ativos de software para serem reutilizados em outras ferramentas do mesmo domínio. Se for possível, quais os passos que devem ser seguidos?

Nesse sentido, a execução desse TCC possibilitou identificar que os domínios não possuem propriedades específicas para alguns atributos dos artefatos. Essa questão de pesquisa pode mapear os atributos mais comuns e/ou compartilhados entre ferramentas diferentes do mesmo domínio, resultando em contribuições para a evolução dos domínios

OSLC.

Qual o nível de necessidade de se estender as especificações de domínios OSLC?

Existem especificações de domínio para diversas fases do ciclo de vida da aplicação. Entretanto, essa pergunta tem como objetivo identificar se é preciso estender estes e em qual nível/frequência. Além disso, tem o propósito de conhecer o que deve ser levado em consideração na extensão dos domínios existentes (funcionalidade das ferramentas, atributos dos artefatos, etc.).

A literatura mostra que existem propostas para a extensão e customização dos domínios OSLC. Ainda, os adaptadores OSLC seguem restritamente os domínios OSLC. No entanto, não existe uma métrica sobre o aprendizado do OSLC em relação ao nível de experiência de profissionais.

ÍNDICE