

PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the TRANSACT consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with the prior written consent of the TRANSACT consortium. This restriction legend shall not be altered or obliterated on or from this document.

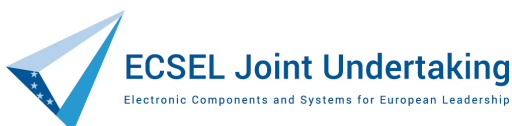


Transform safety-critical Cyber Physical Systems into distributed solutions for end-users and partners

D32 (D6.5)

Open source project proposal

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Finland, Germany, Poland, Austria, Spain, Belgium, Denmark, Norway.



Document Information

Project	TRANSACT
Grant Agreement No.	101007260
Work Package No.	WP6
Task No.	T6.2
Deliverable No.	D32
Deliverable No. in WP	D6.5
Deliverable Title	Open source project proposal
Nature	Report
Dissemination Level	Public
Document Version	v1.0
Date	26/04/2024
Contact	Marco Jahn
Organization	Eclipse Foundation Europe GmbH
Phone	N/A
E-Mail	marco.jahn@eclipse-foundation.org

Authors Table

Name	Company	E-Mail
Marco Jahn	ECL	marco.jahn@eclipse-foundation.org
Javier Fernández-Bravo Peñuela	ITI	fjfernandez@iti.es

Reviewers Table

Version	Date	Reviewer
v0.5	15.04.2024	Wolfram Ratzke, AVL
V0.5	17.04.2024	Mitra Nasri, TUE

Change History

Version	Date	Reason for Change	Affected pages
v0.1	06.09.2023	Initial outline & draft	All
v0.2	10.10.2023	Updated outline after discussing horizontal demonstrator	All
v0.3	24.01.2024	Add chapter on POOSL, open source compliance	All
v0.4	19.02.2024	Add chapter on experimentation environment	All
V0.5	26.03.2024	Complete all chapters, ready for internal review	All
V1.0	26.04.2024	Final version submitted to EC	

Table of Contents

1	GLOSSARY	6
2	INTRODUCTION	7
2.1	ROLE OF THE DELIVERABLE	7
2.2	RELATIONSHIP TO OTHER TRANSACT DOCUMENTS	7
3	PROFESSIONAL, INDUSTRY-GRADE OPEN SOURCE	8
3.1	THE BUSINESS VALUE OF OPEN SOURCE	8
3.2	OPEN SOURCE IN SAFETY- AND SECURITY-RELEVANT DOMAINS: AUTOMOTIVE INDUSTRY	8
3.3	COMMUNITY-DRIVEN OPEN SOURCE	9
3.4	GOVERNANCE	10
3.4.1	<i>The Eclipse Development Process</i>	10
3.4.2	<i>Creating a vendor-neutral, community-driven open-source project at the Eclipse Foundation</i>	10
4	ECLIPSE POOSL	12
4.1	ECLIPSE POOSL OPEN SOURCE PROJECT PROPOSAL	12
4.1.1	<i>Background</i>	12
4.1.2	<i>Scope</i>	12
4.1.3	<i>Description</i>	12
4.1.4	<i>Why Here?</i>	13
4.1.5	<i>Licenses</i>	13
4.1.6	<i>Project Scheduling</i>	13
4.1.7	<i>Future Work</i>	13
4.2	ECLIPSE POOSL SUCCESS STORY IN OPEN SOURCE	13
5	TRANSACT OPEN SOURCE EXPERIMENTATION ENVIRONMENT	14
5.1	ECLIPSE RESEARCH LABS	14
5.2	HORIZONTAL DEMONSTRATOR: A TRANSACT OPEN SOURCE REFERENCE IMPLEMENTATION	14
6	OPEN SOURCE LICENSES AND IP COMPLIANCE	17
6.1.1	<i>Open source license spectrum</i>	17
6.1.2	<i>Open source license compliance and large-scale distributed systems</i>	18
6.1.3	<i>Eclipse Dash License Tool</i>	19
6.1.4	<i>OSS Review Toolkit for continuous license compliance</i>	20
6.1.5	<i>Assessment</i>	25
7	CONCLUSION	26
8	APPENDIX A - REQUIREMENTS FOR A CONTINUOUS COMPLIANCE TOOL	27

List of Figures

Figure 1: Eclipse project creation process	11
Figure 2 POOSL Enterprise Version	13
Figure 3: Eclipse Research Labs.....	14
Figure 4: Ecosystem of open-source technologies for distributed data processing supplied by Radiatus.	15
Figure 5: Mapping of technologies to architectural components in the 3D image reconstruction demonstrator.....	16
Figure 6: The software licenses spectrum.....	18
Figure 7: Vetting third-party content with Dash License.....	20
Figure 8: Overview of ORT installation	22
Figure 9: List of all checked projects.....	23
Figure 10: Main page of an ORT run	24
Figure 11: ORT WebApp report.....	25

List of Tables

Table 1: Terms, Abbreviations and Definitions	6
---	---

1 Glossary

Term	Definition
BDaaS	Big Data as a Service
CLI	Command Line Interface
CPS	Cyber-physical System
CSV	Comma separated values
EDP	Eclipse Development Process
EMO	Eclipse Management Organisation
IoT	Internet of Things
IP	Intellectual property
ORT	OSS Review Toolkit
OS	Open source
OSS	Open-source software
POOSL	Parallel Object-Oriented Specification Language
SCDCPS	Safety-critical distributed cyber-physical systems
SDV	Software-Defined Vehicle

Table 1: Terms, Abbreviations and Definitions

2 Introduction

This deliverable summarizes the open-source perspective on the TRANSACT project.

TRANSACT is an industry-driven project in the domain of safe and secure cyber-physical systems, a domain that is characterized by closed, proprietary software and high demands and strict requirements on certification and safety and security assurance. Traditionally, aspects that don't foster the adoption and development of open source tools per se.

In TRANSACT several approaches were investigated to identify potential directions for leveraging the benefits of open source:

- Eclipse POOSL as an example of professional, vendor-neutral, community-driven open source project
- A horizontal demonstrator implementing the TRANSACT reference architecture in open source, allowing experimentation in a publicly available open source environment
- An experimental setup of an open source compliance tool for assessing open-source license compliance in a large, multi-project environment

2.1 Role of the deliverable

The purpose of this deliverable is to:

- Introduce the idea, benefits, and characteristics of vendor-neutral, community-driven open source in TRANSACT domains, which are characterized by high safety and security requirements (Chapter 3)
- Exemplify successful, industry-grade open source with the Eclipse POOSL project (Chapter 4)
- Present the open source experimentation environment hosting the horizontal demonstrator as an open-source reference implementation of the TRANSACT architecture (Chapter 5)
- Report on the experiments for large-scale, open source license compliance (Chapter 6)

2.2 Relationship to other TRANSACT documents

This document relates to the following TRANSACT deliverables:

- D26 (D2.4) Reference architectures for SCDCPS v2
- D31 (D5.2) TRANSACT basic horizontal demonstrator

3 Professional, industry-grade open source

3.1 The business value of open source

Open source software accounts for 80 to 90 percent of the code in a typical product, service, or application¹. The remaining 10 to 20 percent come from businesses that adopt open source and use it as a foundation to differentiate and add value for their customers. These high adoption rates confirm that open source reduces costs, accelerates time to market, lowers risks, and increases opportunities for innovation.

Some of the biggest winners in our software-centric world are internet platform companies that deliver an ecosystem of solutions to consumers. According to digital economist Dr. Holger Schmidt, the top 100 platform companies in the world were valued at a total of 15.5 trillion dollars in July 2021². The business models that platform economy leaders rely on are only possible with open source software. These companies could not have reached their current scale by purchasing or developing all of the software required to build and run their businesses.

Many of these organizations also recognize the business benefits of actively participating in open source software projects. They understand that an effective open source strategy goes well beyond simply consuming available software. To realize the full business potential of open source, it's crucial to engage with, support, and contribute to the communities that produce the software.

In his study, "Learning by Contributing: Gaining Competitive Advantage Through Contribution to Crowdsourced Public Goods," Frank Nagle of Harvard Business School found that companies that contribute and give back to the community learn how to better use the open source software in their own environment, creating a competitive advantage. According to Nagle, companies that pay employees to contribute to open source software can boost company productivity by up to 100 percent, compared with companies that simply use or consume the software. They can also improve their image and their ability to recruit top talent. "As the technical world is increasingly open source and everybody can use the same types of technology, gaining these kinds of edges and increasing your competitive advantage is pretty important," says Nagle.

3.2 Open source in safety- and security-relevant domains: automotive industry

Looking at the automotive industry as an example: the first cars were mechanical masterpieces, but didn't include anything that even resembles an integrated circuit. As the industry evolved, cars began to leverage advances in electrical engineering and became more technology-driven. Today, most cars are essentially computers on wheels. They rely on software running on hundreds of embedded sensors and actuators to control everything from steering, safety, and fuel systems to advanced touch screens and infotainment systems. By most estimates, there are already at least 100 million lines of code in most modern, non-autonomous cars. For comparison, the flight software in a Boeing 787 aircraft requires approximately 14 million lines of code. As the industry shifts to electric and autonomous vehicles, it's expected that cars will essentially become servers on wheels with 300 to 500 million lines of code. This massive volume of complex code is essential for safe and reliable vehicle operation, and to support the software features and functionality that have become key competitive differentiators.

¹ https://www.sonatype.com/hubfs/SSC/Software_Supply_Chain_Inforgraphic.pdf?t=1468857601884

² <https://www.platformeconomy.io/blog/value-of-the-top-100-platform-rises-to-15-5-trillion>

A future SDV (Software-Defined Vehicle) platform is likely to consist of multiple and exchangeable building blocks, incorporating both open source and proprietary elements supported by single or multiple vendors. To enable a modular SDV platform with exchangeable building blocks, there is a need for standardised interfaces and abstractions, a development that is actively underway through various industry collaborations.

The automotive industry is just one example. The shift from an electro-mechanical focus to special-purpose embedded computers to software-driven solutions is found in almost every technology and tool is used at home and at work. Open source software is the common denominator across all of these solutions.

3.3 Community-driven open source

Two prominent approaches to open source are single-vendor and community-driven open source. Understanding the distinctions between these models is crucial for navigating the landscape of open source projects.

Single-vendor open source projects are led by a single organization, with contributions mainly from its employees. Decision-making is centralized, and resources are allocated by the sponsoring company. In contrast, community-driven open source projects have decentralized development, with contributions from diverse individuals and organizations. Governance is meritocratic, and project direction is influenced by community consensus.

Open source foundations enable vendor-neutral, community-driven open source, which can bring various advantages also in terms of commercial innovation and adoption:

Communities reduce risks: As an example, looking at Eclipse IoT³, with more than 45 IoT projects across device, gateway, cloud, security, edge, and other domains, organizations have easy access to all of the building blocks needed to develop end-to-end IoT solutions. With numerous global leaders in IoT technologies actively contributing to, and adopting, the open source code, all community members can be assured they're incorporating robust and reliable IoT technologies that reflect industry requirements (a mapping of these technologies to the TRANSACT reference architecture can also be found in D26 (D2.4) Reference architectures for distributed safety-critical distributed cyber-physical systems v2).

Longevity is assured: Widely adopted open source software is continuously enhanced by developers with a variety of requirements and skill sets. As a result, the software remains relevant and usable over the long term. In openly governed projects, there is no single vendor who can choose to end support for it, or take it in a specific direction. The functionality evolves in a structured way, according to community needs and interests.

Flexibility is built-in: Open source software based on open specifications enables even higher levels of interoperability and innovation. Organizations can drive broad adoption of the technologies in which they invest. And they have the architectural flexibility to easily leverage faster and more advanced technologies with no need to rewrite their applications. It also keeps competitive pressure high, and is a major factor in an organization's ability to quickly enhance its offerings to meet modern demands at scale.

Open innovation reaches industrial scale: Community-driven open source is proven to be the best way to co-innovate on high-quality, scalable, and sustainable technologies that allow organizations to build better products faster, accelerate revenues, and create value. Collaborating with competitors on non-differentiating technologies frees scarce resources to focus on delivering value-added and differentiating features faster.

³ <https://iot.eclipse.org/>

It's also quickly becoming a leading approach to create new platforms that drive mainstream adoption of new technologies such as machine learning, edge computing, and automotive technologies.

Value grows with level of involvement: As organizations evolve from users to contributors to leaders of open source projects, and become more involved in open source communities, the benefits they derive from their involvement grow well beyond traditional engineering functions. These organizations have greater ability to translate their open innovation, enhanced capabilities, and open source culture into customer value that results in profitable growth

3.4 Governance

*"In open source software projects, the rules and customs that define who gets to do what (and how they are supposed to do it) is called a project's governance model."*⁴ In community-driven open source projects, governance is key because these projects are all about collaboration between different organisations, companies, and individuals. Open source foundations provide such kind of governance for community-driven open source projects acting as a neutral third-party and establishing an environment for open collaboration.

3.4.1 The Eclipse Development Process

The Eclipse Foundation has laid out these rules and customs in the Eclipse Development Process⁵ (EDP): The EDP describes the manner in which community-driven open source software is governed at the Eclipse Foundation. The EDP does not prescribe any particular development methodology; it is more concerned with the larger-scale aspects of open source project life cycle, including such things as reviews, processes for running votes and elections, bringing new committers onto a project, etc.

Four basic principles (aka the "open source rules of engagement") lie at the heart of the EDP:

- **Transparency:** a project's discussions, minutes, deliberations, project plans, plans for new features, and other artifacts are open, public, and easily accessible.
- **Openness:** the project is open to all. Everyone participates with the same rules; there are no rules to exclude any potential contributors which include, of course, direct competitors in the marketplace.
- **Meritocracy:** the more that somebody contributes, the more responsibility they will earn. Leadership roles are merit-based and earned by peer acclaim.
- **Vendor-neutrality:** maintaining a level playing field. No vendor is permitted to dominate a project, and nobody can be excluded from participating in a project based on their employment status.

Another important aspect is intellectual property cleanliness: code produced by an Eclipse project is used by organisations to build products. These adopters of Eclipse technology need to have some assurance that the IP they're basing their products on is *clean*: the organisation or individuals who claim copyright of the code are the legitimate copyright holders, and the copyright holders legitimately agree to make the code available under the license(s) that the project works under.

3.4.2 Creating a vendor-neutral, community-driven open-source project at the Eclipse Foundation

The process of creating a project at the Eclipse Foundation follows a strict process (see Figure 1) to implement the aforementioned open source rules of engagement.

⁴ <https://www.redhat.com/en/blog/understanding-open-source-governance-models>

⁵ https://www.eclipse.org/projects/dev_process/

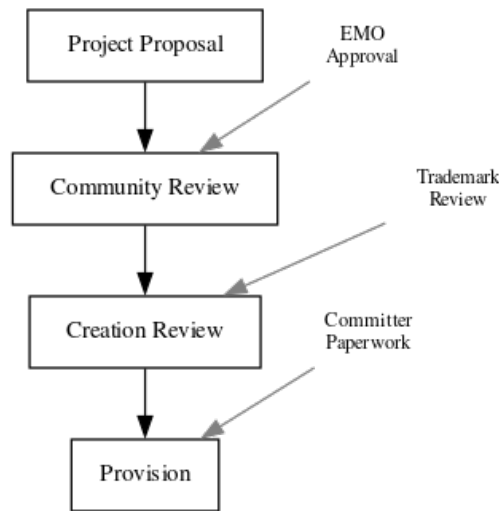


Figure 1: Eclipse project creation process

Eclipse open source projects start with a *proposal* that is made available to the community for review. During that phase, which is called *community review*, everybody in the Eclipse open source community can view the proposal and provide feedback. A proposal must minimally include a description of the project, a declaration of scope, and a list of prospective members (project leads and committers) before it is made accessible to the public for *community review*.

At the end of the *community review* period, the project goes into *creation review*. Creation reviews are an extension of, and a natural conclusion to, the community review period for a project proposal. As such, the commitment during the review is to continue to monitor the proposal's communication channel and answer any questions related to the project proposal. After this phase, which lasts for 5 business days, the project resources are *provisioned*, i.e., the project's source code can be committed to the respective Eclipse repository.

After the project code is pushed into the project repository, the project team can create and distribute *milestone* builds for the first release. However, all intellectual property must be approved by the Eclipse IP Team before the project team can issue any official releases. Chapter 6 provides further details on the topics of IP and license compliance.

The following chapter describes project proposal of Eclipse POOSL.

4 Eclipse POOSL

Eclipse POOSL was created as an Eclipse Foundation project in October 2021 following the governance and project creation rules as described in Chapter 3. The following sections are the project proposal that was committed to the Eclipse Foundation.

4.1 Eclipse POOSL Open Source Project Proposal

4.1.1 Background

High-tech (embedded) systems are characterized by hardware and software components interacting closely with each other. The complexity of these systems requires modelling methods that structure and support the design process. In the early phases of system development, requirements are still unclear and many decisions have to be taken on the system architecture, the responsibilities, the behaviour and the interactions between components. For these early phases, there is a lack of tools combining modelling with analysis at the appropriate abstraction level.

4.1.2 Scope

Eclipse POOSL (Parallel Object-Oriented Specification Language) provides tools for the editing (both textual and graphical) and debugging (using the Rotalumis simulator) of POOSL models. Debugging depends on the Rotalumis⁶ simulator from Eindhoven University of Technology that is out-of-scope.

4.1.3 Description

Eclipse POOSL (Parallel Object-Oriented Specification Language)⁷ and the accompanying tools offer a general purpose method for describing and simulating (embedded) system architectures for the early evaluation of key structural and behavioural concepts, requirements and performance. This lightweight modelling and simulation approach shortens the development time of complex high-tech systems by providing fast insights into requirements and early design decisions, thereby reducing the risk of expensive iterations during design, integration and testing.

POOSL targets discrete domains with a notion of time. It provides an intuitive syntax that matches well-spread approaches such as component-based software development. The semantics is based on formal techniques, which ensures unambiguous simulation of models, functional verification and performance analysis.

POOSL is accompanied with user-friendly industry-strength Eclipse-based tools with strong focus on interactive model development and scalable analysis. Light-weight model development and validation is supported by an Integrated Development Environment in combination with a scalable simulator. The tools allow easy integration with external visualization and control tools via sockets and files. Various model libraries provide common data structures, stochastic distributions and observers for evaluation of, for example, worst/best case and average case performance properties.

POOSL was originally developed at Eindhoven University of Technology (TU/e) and was adopted by ESI (TNO) as a standard modelling and analysis tool following its successful application in the high-tech industry.

⁶ <https://www.es.ele.tue.nl/poosl/Tools/rotalumis/>

⁷ <https://www.es.ele.tue.nl/premadona/publications/TFGHPV07.pdf>

4.1.4 Why Here?

Eclipse POOSL is based on several Eclipse modelling projects: EMF, Sirius, Xtext. In addition it is based on the Eclipse Debug project. The genericity of the POOSL language makes it suitable for modelling and analysing many concurrent hardware/software systems.

4.1.5 Licenses

Eclipse Public License 2.0

4.1.6 Project Scheduling

The initial contribution is ready to be submitted once the proposal passes.

Incubation release mid-October 2021.

Official release end of December 2021.

4.1.7 Future Work

Bugfixes, UI & performance improvements.

Integrate Eclipse TRACE4CPS⁸

4.2 Eclipse POOSL success story in open source

Eclipse POOSL is applied and further developed in the scope of TRANSACT, as explained in several deliverables. Being a community-driven, open source project, it comes with all the potential advantages explained before. In fact, Eclipse POOSL is being adopted by the company OBEO for commercial up-take. The company offers professional support and an OBEO enterprise version⁹ is available (see Figure 2).



Figure 2 POOSL Enterprise Version

⁸<https://projects.eclipse.org/proposals/eclipse-trace4cps>

⁹<https://www.obeosoftware.com/en/obeo-enterprise-for-pool>

Version	Nature / Level	Date	Page
v1.0	R / PU	26/04/2024	13 of 29

5 TRANSACT open source experimentation environment

Since the TRANSACT use cases very much focus on industrial applications, we also provide an experimentation environment for developing TRANSACT solutions in open source. This environment contains the TRANSACT horizontal demonstrator, which acts as a purely open source implementation of the TRANSACT architecture, allowing everybody to inspect, use, and extend the implemented concepts.

5.1 Eclipse Research Labs

Eclipse Research Labs is the GitLab environment hosting the open source code of the horizontal demonstrator. This environment allows to provide experimental open source results without having to implement the full and strict Eclipse Development Process as described in Section 3.4.1. Nevertheless, it still introduces the concepts of openness and transparency and allows to already introduce IP-compliance aspects. It is also a place to share open source results across research projects and as such a tool to communicate. Figure 3 shows a snapshot of the Eclipse Research Labs and groups for different research projects including TRANSACT.

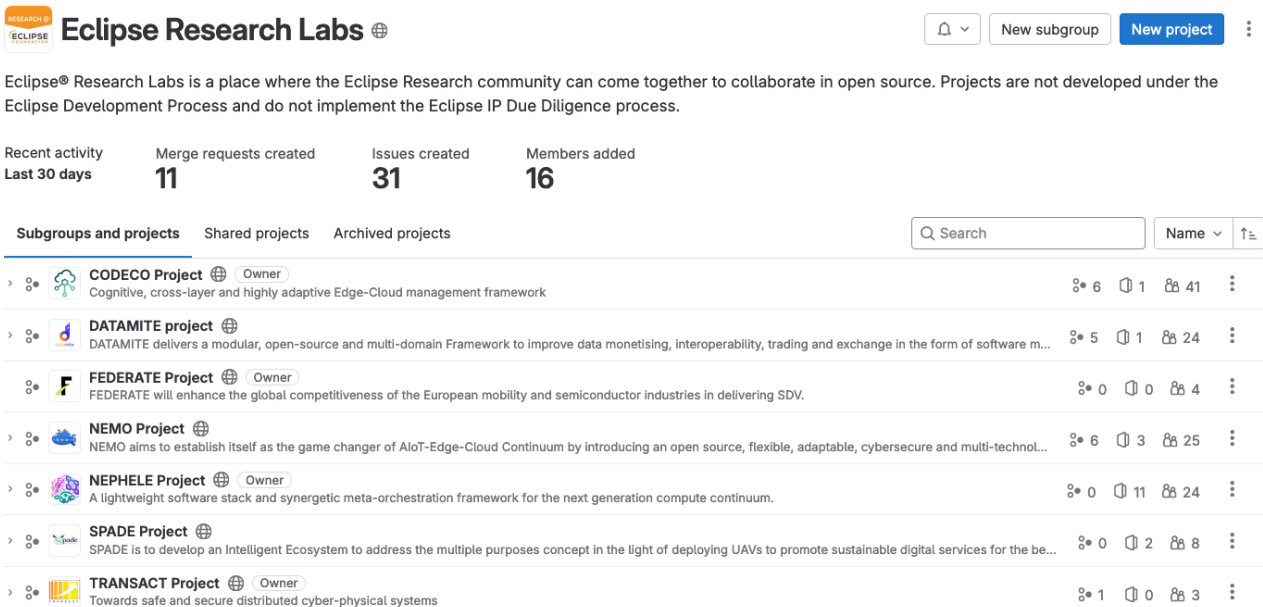


Figure 3: Eclipse Research Labs

5.2 Horizontal demonstrator: A TRANSACT open source reference implementation

TRANSACT’s horizontal demonstrator is an experimental platform for the deployment of solutions, scenarios and technologies to showcase the outcomes of the TRANSACT project. It provides, as well, a setting to work on scenarios based on concepts, solutions, methods and architectural components not covered by the industrial use cases.

Version	Nature / Level	Date	Page
v1.0	R / PU	26/04/2024	14 of 29

This platform comprises two clusters (cloud and device), both running Kumori Platform¹⁰, a PaaS based on Kubernetes where users can deploy service applications, by specifying a group of microservices and how they connect to each other. In addition, the cloud cluster includes a Radiatus deployment on top of Kumori. Radiatus is a Big Data as a Service (BDaaS) platform developed by ITI that operates as a technological enabler for the provision and management of several open-source tools and technologies related to distributed data processing. Next, Figure 4 displays the technological ecosystem currently supplied by Radiatus.

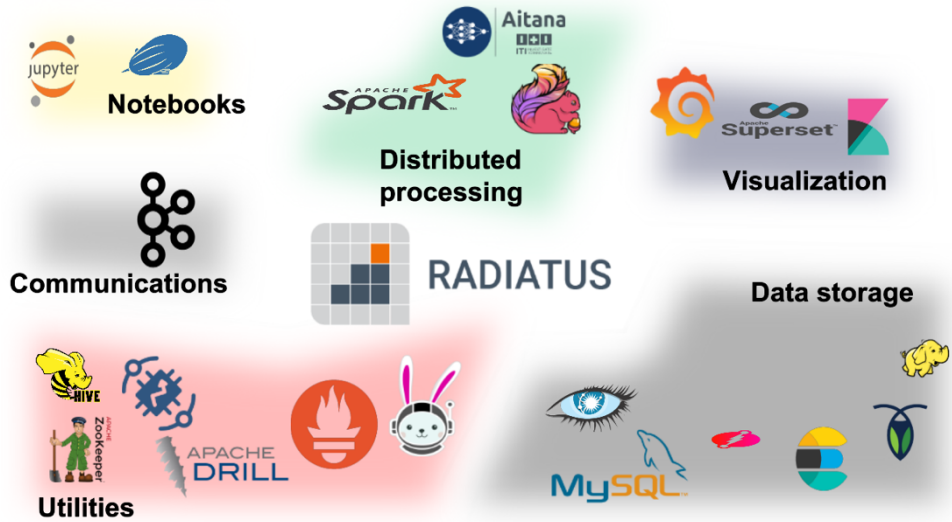


Figure 4: Ecosystem of open-source technologies for distributed data processing supplied by Radiatus.

By providing this deployment setting, the horizontal demonstrator establishes the foundations to build implementations of TRANSACT’s reference architecture that cover fresh scenarios via open-source technologies. To ensure that an implementation sticks to the reference architecture, it must encompass at least two computing tiers in the compute continuum. In addition, the (open-source) technologies, solutions, applications and algorithms deployed need to be mappable to architectural components and building blocks.

TNO has contributed their software to this platform, featuring an open-source demonstrator for 3D image reconstruction, which runs on top of Kumori Platform and whose cloud layer consumes software services provided by Radiatus. Figure 5 below depicts the approach adopted for this setting, based on an instantiation of the reference architecture.

¹⁰ Kumori Platform Community, <https://gitlab.com/kumori-systems/community/>

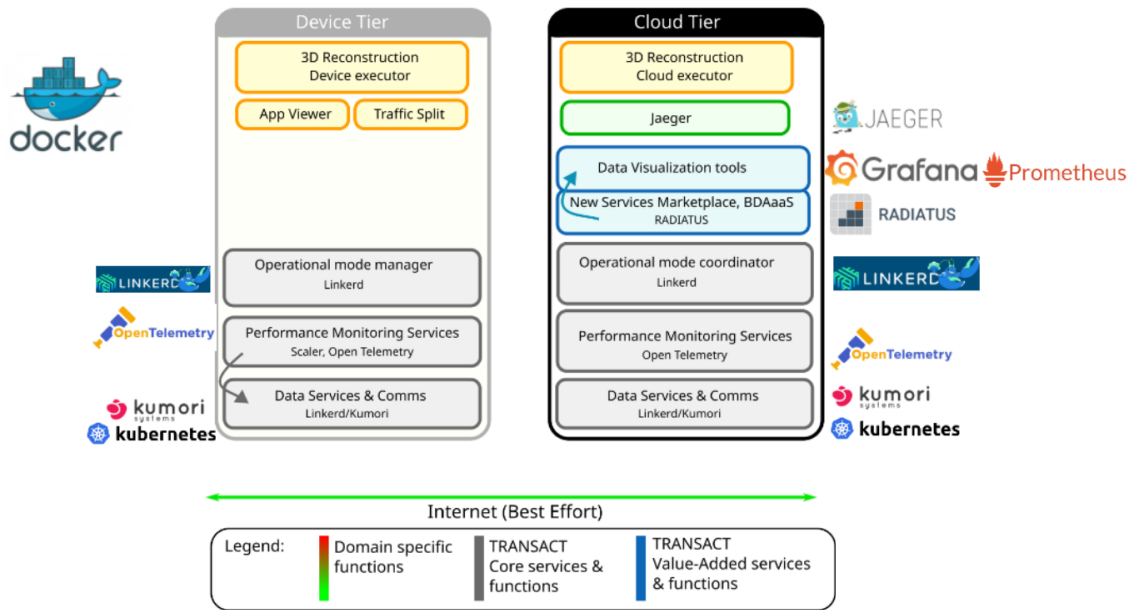


Figure 5: Mapping of technologies to architectural components in the 3D image reconstruction demonstrator.

This demonstration spans the device and cloud tiers of the reference architecture, assimilating open-source technologies that are matched to specific architectural components. Application and domain specific functions are enclosed in Docker containers that run on Kumori (based on Kubernetes). Value-added services and functions are provided by technologies (Grafana, Prometheus, Jaeger) that run on Radiatus. Core services and functions are implemented through Open Telemetry and Linkerd, that run directly on top of Kumori.

By heeding this approach as an example model, further domain specific applications that rely on open-source software and consume services provided by open-source technologies can be adapted for their deployment on a distributed setting. This aims to display how TRANSACT’s reference architecture can accommodate novel scenarios that, by combining different open-source technological options, are realized into running implementations.

The full technical in-depth description of the horizontal demonstrator can be found in D31 (D5.2) TRANSACT basic horizontal demonstrator.

6 Open source licenses and IP compliance

This chapter provides an introduction to open source license compliance and reports on an experimental deployment to test license compliance processes for managing large set of projects.

6.1.1 Open source license spectrum

Open source licenses give adopters and developers permission to use, modify, and distribute the open source software freely, under the terms laid out in the specific license. The Open Source Initiative¹¹ provides a very good definition of Open Source Software (OSS) and defines it in *10 commandments*¹²:

1. **Free redistribution:** The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. **Include source code:** The program must include source code, and must allow distribution in source code as well as compiled form.
3. **Modifications and derived works:** The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of author's source code:** The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5. **No discrimination against person and groups:** The license must not discriminate against any person or group of persons.
6. **No discrimination against fields of endeavour:** The license must not restrict anyone from making use of the program in a specific field of endeavour.
7. **Distribution of license:** The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License not specific to a product:** The rights attached to the program must not depend on the program's being part of a particular software distribution.
9. **License not restricting other software:** The license must not place restrictions on other software that is distributed along with the licensed software.
10. **License technology neutral:** No provision of the license may be predicated on any individual technology or style of interface.

There is a huge variety of open source licenses, with different constraints and compatibility. Some licenses can be mixed, while others are not compatible together; it also depends on the type of usage (e.g., sources included in the build, or simply downloaded as binaries).

As an example, some licenses can work with others in 3rd party dependencies (e.g., it is ok to add a LGPL dependency in an EPL project), while others simply cannot: it is not ok to add a GPL dependency since GPL

¹¹ <https://opensource.org>

¹² <https://opensource.org/osd-annotated>

forbids it. The license spectrum ranges from permissive licenses (e.g., MIT, BSD, Apache) to proprietary licenses which typically don't allow modification or distribution of the software (see Figure 6).

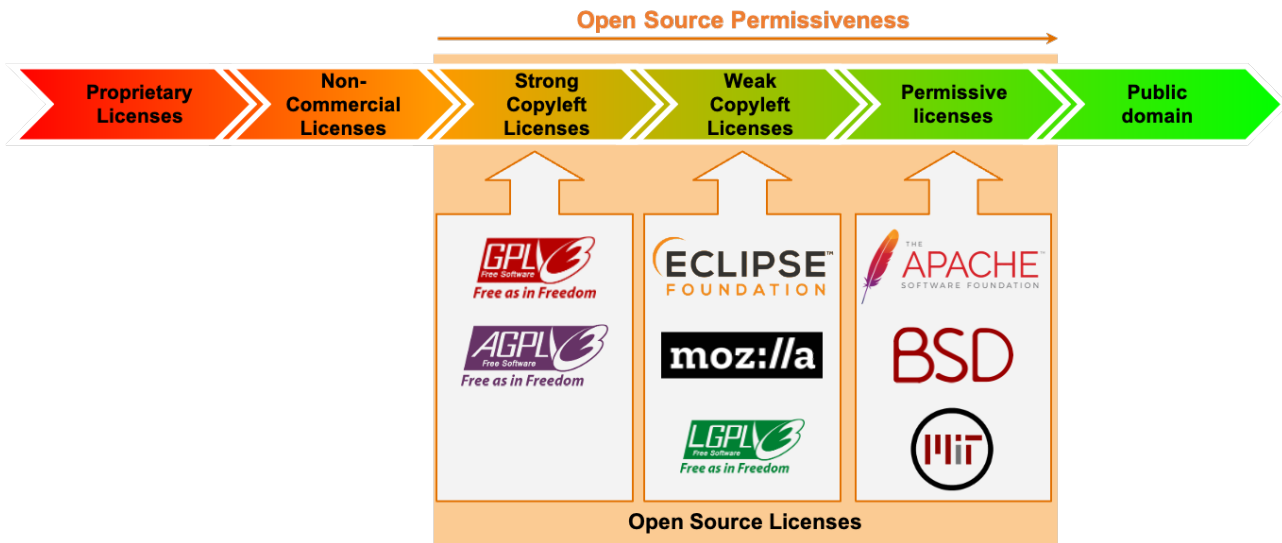


Figure 6: The software licenses spectrum

In between, there are the “Copyleft Licenses”. In contrast to permissive licenses, these are considered protective or reciprocal as they impose more constraints on the users or integrators of the software. Within this share of the spectrum, we find both strong (e.g., GPL, AGPL) and weak (e.g., EPL, MPL) copyleft licenses. Both allow free distribution and modification of the software. The devil is in the details: using strong copyleft licensed code usually forces the licensee to put their own code under that same license whereas weak copyleft requires to only publish changes to the original code under the original license. In other words, weak copyleft licenses allow free distribution and modification of the software (also in proprietary products) but require that changes made to the original code stay under the original license. Thus, weak copyleft licenses foster collaboration and innovation by ensuring that improvements to the open source project stay open while still allowing its use in commercial products or providing added-value services on top.

6.1.2 Open source license compliance and large-scale distributed systems

When using or extending open-source software one has to adhere to the terms laid out by the particular license. That is where open-source license compliance comes into play. License compliance in software projects is a complex problem and involves several aspects which bring a lot of uncertainty, e.g., availability of license information in structured format or grey areas in the assessment of compatibility between different licenses. This is even more the case when considering the strict requirements of safety- and security-sensitive domains.

When software becomes increasingly distributed also license compliance becomes more complex. Every piece of software we rely on may come with dependencies each of which may come with dependencies and so on and so forth. Keeping an overview of the state of license compliance of large-scale distributed software systems is also not an easy task. Identifying, and fetching all dependencies, can be complex. Each project may use different build tools, with different formats to store the dependencies. Each dependency in turn can use a different technology and build tool, and so recursively. And modern software projects have a lot of dependencies. So clearly, tool-support is needed to perform license compliance checks. A variety of tools exist from very simple ones to powerful frameworks covering many aspects. Currently, at the Eclipse Foundation we provide a command line tool to committers which allows them to vet the third-party content of a project. This Eclipse Dash License Tool is available under EPL-2.0 license and described in Section 6.1.3.

Version	Nature / Level	Date	Page
v1.0	R / PU	26/04/2024	18 of 29

Such tool, typically requires pro-active behaviour by the committer/developer. He has to run the tool or integrate it into his project(s)' build processes. In the scope of TRANSACT a different approach was tested, i.e., setting up a more powerful tool to keep track of the IP issues of many projects. This setup is described in Section 6.1.4.

6.1.3 Eclipse Dash License Tool

The Eclipse Dash License Tool¹³ (short: Dash License) is a command line tool that can be used by developers (or anybody else) to gain information about the third part licenses in their open source project. The tool itself does not identify dependencies (at least not in general). Rather, the value it provides starts after the list of dependencies are identified by build tools. That is, the tool works on the list of dependencies with which it is provided.

The tool is only as good as the input with which it is provided, and it is up to the committer to ensure that the input provided is correct. That means, dependencies that are not automatically discovered by build tools must be vetted manually. The CLI accepts a flat file with each line containing a content identifier (ClearlyDefined id, Maven coordinates, or NPM identifier); it also supports a small number of file formats including package-lock.json or yarn.lock files. A Maven plugin that is capable of processing a dependency list extracted from a pom.xml file is also provided.

It can generate a file that contains CSV content with one line for each line of input, mapping a package to a license along with whether that content is approved for use by an Eclipse project or restricted, meaning that this content would need to be inspected further.

The current implementation uses two sources for license information. The first source is an Eclipse Foundation service that leverages data that the Eclipse Foundation's IP Team has collected over the years (and continues to collect). When that source does not have information for a piece of content, ClearlyDefined's service is used.

The idea was to have some code that can be used to check the licenses of content, but write it in a manner that would make it easy to generate, for example, a Maven plug-in. The main focus, however, has been making this work as a CLI so that it can be used to sort out licenses for Maven, package-lock.json, yarn.lock, etc.

Figure 7 shows how the process of vetting third-party content with the Dash License Tool. Once the list of dependencies has been created in the proper format, the tool is very straightforward. It takes the file, checks each line against its sources (i.e. the Eclipse DB and ClearlyDefined) and provides the respective output, i.e. if a library is approved or needs further investigation.

¹³ <https://github.com/eclipse/dash-licenses>

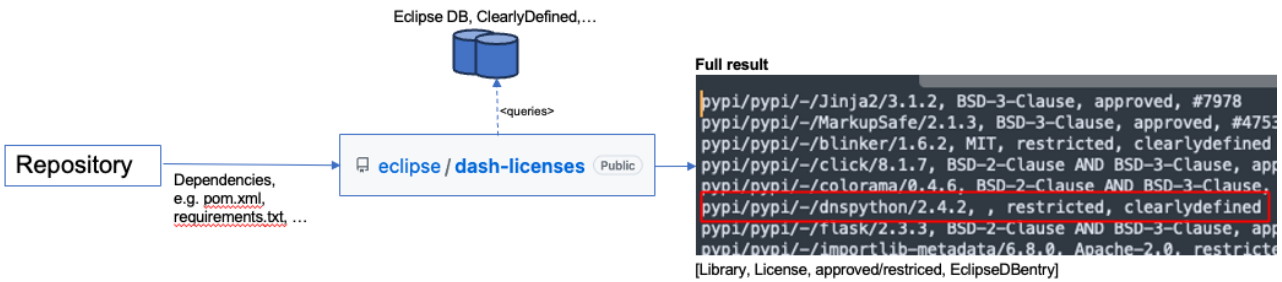


Figure 7: Vetting third-party content with Dash License

6.1.4 OSS Review Toolkit for continuous license compliance

In the scope of TRANSACT the applicability of ORT¹⁴ (OSS Review Toolkit) was tested to find out if a more feature-rich framework (compared to the Dash License Tool) can be employed to continuously perform license compliance checks for a larger number of projects/repositories.

6.1.4.1 Requirements

For the experiment, we identified the following requirements (based on expert interviews with the Eclipse Management Organisation Team):

Functional requirements:

- List licenses and IP in project code and third-party dependencies.
- Check license compatibility (according to Eclipse's rules) and display those that cannot be automatically validated.
- Reuse wisdom from IPZilla.
- Generate Bill of Materials.
- Ease the work of the legal department and automate as much as possible the IP checking process, leaving only warnings/errors when unsure. When correct, provide a link to the justification.

Non-functional requirements:

- Do not interfere with existing code or build processes: simply clone code and run the analysis.
- Be technology-agnostic, regarding both the code and the build system.
- Rely on standards (like SPDX) and trusted external data sources (like clearlydefined.io, and the Eclipse Foundation's own collection of intellectual property data) as much as possible.
- Can be easily executed on any repository (i.e. have a simple script to start the whole process).

The detailed list of these requirements can be found in Appendix A - Requirements for a continuous compliance tool.

¹⁴ <https://oss-review-toolkit.org/ort/>

The OSS Review Toolkit was chosen as test candidate because it provides the following features:

- It can use different scanners, allowing us to try and select the best for our usage: ScanCode from NexB is the default for scanning, FOSSID¹⁵, Askalono¹⁶, lc¹⁷, Licensee¹⁸, SCANOSS¹⁹
- It provides a resolution mechanism to describe why some violations are accepted: resolutions.yml. This helps fix corner-cases and exceptions – when hosting a large amount of open source projects with different technologies, domains and constraints, exceptions are plenty.
- It provides a mechanism to manage missing / incomplete package metadata: curations.yml. This is required to implement our own IP knowledge base and reuse efficiently the years of IP clearance done by the foundation’s lawyers.
- It provides a mechanism for licenses categorisation and custom policies. This is required in the case of the Eclipse Foundation to implement the custom IP Policy, which differs from one organisation to the other.
- ORT has several publishers, including Bill of Materials, Notices, and a webapp. This enables us to:
 - Automatically generate the due documentation (notices) to be legally compliant.
 - Provide different ways to display the interesting data, depending on people and projects. Some prefer a text file, while some others will go for the feature-rich WebApp HTML export.
 - Anticipate security-related questions through SBOMs. Recent events in the field, like the SolarWind²⁰ hack, demonstrate how the supply chain is becoming increasingly necessary.

It should be noted that other tools like Fossology²¹ and SW360²² have a different scope, and are more oriented towards Software Composition Analysis. The IP & License checking stands as an added feature, with less abilities than a dedicated tool like ORT.

6.1.4.2 Setup and configuration

In the following we describe the initial setup of our ORT instance. We have scripts to build our custom list of licenses, and to rebuild regularly our knowledge base from our different sources (Eclipse’s own IP knowledge base, plus other instances like clearlydefined²³). We then execute ORT with this configuration, and publish its results alongside a custom, dedicated static website.

¹⁵ <https://fossid.com>

¹⁶ <https://github.com/amzn/askalono>

¹⁷ <https://github.com/boyter/lc>

¹⁸ <https://github.com/licensee/licensee>

¹⁹ <https://www.scanoss.com>

²⁰ <https://www.techtarget.com/whatis/feature/SolarWinds-hack-explained-Everything-you-need-to-know>

²¹ <https://www.fossology.org/>

²² <https://github.com/eclipse-sw360>

²³ <https://clearlydefined.io>

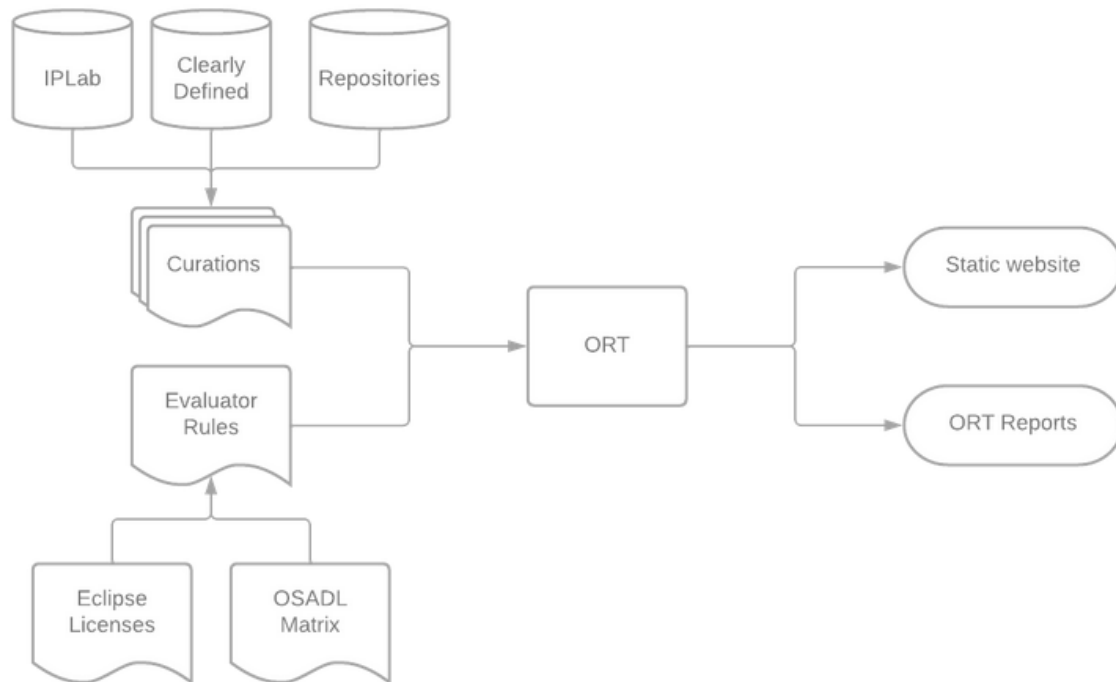


Figure 8: Overview of ORT installation

Figure 8 shows the overview of the ORT installation.

Curations correct invalid or missing package metadata, configure a licenses mapping, and set the concluded license for packages. We've conducted a few tests on our instance and eventually settled down on this format:

```
-id: "NPM::argparse:1.0.10"
curations:
  comment: "Manually checked issue, it is a false-positive. License
available at https://www.npmjs.com/package/argparse/v/1.0.10"
  concluded_license: "MIT"
```

As of now curations are automatically exported from the IPZilla API thanks to a new endpoint:

- <http://www.eclipse.org/projects/services/curations.yml.php>

Evaluator rules allow to define precisely what licenses are approved in each situation. They are written in kotlin, thus enabling a wide range of different executions depending on the many possible cases. This wisdom is captured into the evaluator.rules.kts file, publicly available in our repository:

- <https://gitlab.eclipse.org/eclipsefdn/emo-team/eclipsefdn-ort/-/blob/main/conf/evaluator.rules.kts>

6.1.4.3 Web interface & reports

ORT generates a number of reports for different usages. However, we wanted to provide a way to quickly identify important violations for each project and repository. Therefore, we wrote a script to automatically generate a static website from all the runs executed on projects, to allow users to quickly go to their projects and repositories and navigate the various outputs and generated reports. We also created a view on the errors and violations at a global level, to help us tackle the most recurring issues.

The static website is automatically published after each run. Figure 9 is the landing page and shows the list of all IP-checked projects and a timestamp of the latest run per project.

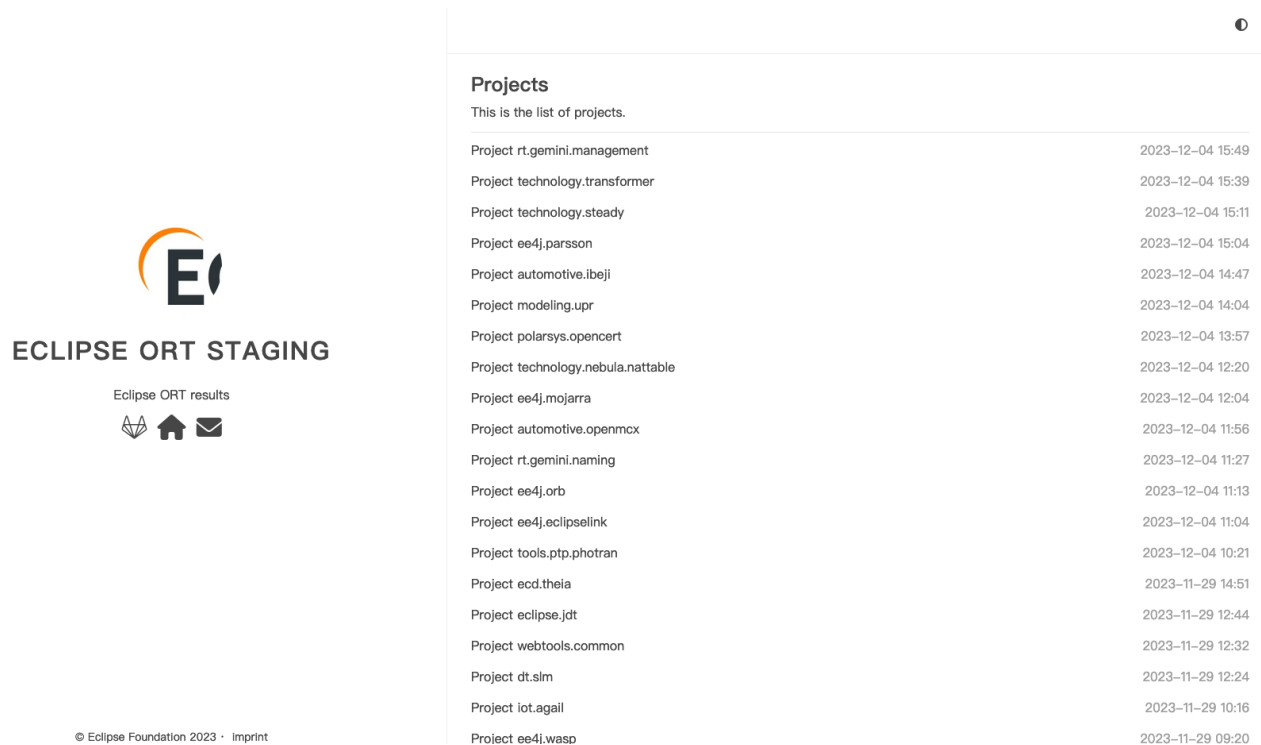




Figure 9: List of all checked projects



ECLIPSE ORT STAGING

Eclipse ORT results



RUN MODELING.POOSL – POOSL – 202304061937

SUMMARY

- Project: [modeling.poosl](#).
- Repository: [poosl](#).
- Date: 2023-04-06T19:37:00.
- Violations: 53 – [Download CSV](#).
 - PROJECT_LICENSE_CHECK 4
 - DEPENDENCY_LICENSE_CHECK 9
 - OSADL_PROJECT_LICENSE_COMPATIBILITY 40.
- Log file: [modeling.poosl_poosl_202304061937.log](#).
- Evaluation result file: [JSON report](#)

REPORTS

List of ORT published reports:

- [WebApp report](#)
- [Static html report](#)
- [Notice file](#)
- [SPDX SBOM](#)
- [CycloneDX SBOM](#)

Figure 10: Main page of an ORT run

Figure 10 shows the overview page of an ORT scan for the Eclipse POOSL project. It allows the user to view different results, such as the license compliance report in WebApp or static format or SBOMs in different formats. When going to the WebApp report the user can have a closer look at the dependencies and potential compliance issues. It should be noted that the violations indicated in Figure 11 are mainly due to further manual configuration that needs to be done.

Summary Table Tree

- Scanned revision 756a21c245eff172e6537c533c2519122235792d of Git repository <https://github.com/eclipse/poosl.git>
- Found 32 files defining 42 unique dependencies within 2 scopes and 2 dependency levels
- Detected 14 licenses and 6 declared licenses
- Completed scan with 25 unresolved policy violations

Rule Violations (25/53) Vulnerabilities (80) Declared Licenses (6) Detected Licenses (14)

Package	Rule	Message
Maven:ch.qos.logback:logback-classic:1.1.2	OSADL_PROJECT_LICENSE_COMPATIBILITY	The outbound license Apache-2.0 of project 'Maven:com.example.org:json-rpc-example:0.1-SNAPSHOT' is incompatible with the inbound license EPL-1.0 of its dependency 'Maven:ch.qos.logback:logback-classic:1.1.2'. Software under a copyleft license such as the EPL-1.0 license normally cannot be redistributed under a non-copyleft license such as the Apache-2.0 license, except if it were explicitly permitted in the licenses.
Maven:ch.qos.logback:logback-core:1.1.2	OSADL_PROJECT_LICENSE_COMPATIBILITY	The outbound license Apache-2.0 of project 'Maven:com.example.org:json-rpc-example:0.1-SNAPSHOT' is incompatible with the inbound license EPL-1.0 of its dependency 'Maven:ch.qos.logback:logback-core:1.1.2'. Software under a copyleft license such as the EPL-1.0 license normally cannot be redistributed under a non-copyleft license such as the Apache-2.0 license, except if it were explicitly permitted in the licenses.
		The outbound license EPL-2.0 of project 'Maven:com.example.org:json-rpc-example:0.1-SNAPSHOT' is incompatible with the inbound license GPL-1.0-or-later of its dependency 'Maven:net.iq.harrier:base64:2.3.9'. Software under a

Figure 11: ORT WebApp report

6.1.5 Assessment

At the end of the prototyping stage, we come to the conclusion that a feature-rich tool for continuous license-compliance checks such as ORT has the potential to ease the task of vetting third-party content for a large number of projects and thus being applicable to large-scale distributed systems. ORT itself met most of the requirements to a sufficient level. The tool allows to get a comprehensive overview of the state of third-party license compliance from a general overview down to project-specific level. The visual interface helps to get a good overview and dig deeper into specific issues. Once set up correctly, runs were triggered automatically without the need for manual intervention. It was also able to create reports as well as SBOMs.

Nevertheless, at the end of the prototyping stage of this experiment it became clear that substantial effort would be required to turn the experiment into a long-lasting, maintainable and stable installation. This goes for maintaining the installation but also for handling and tweaking the rules for the myriad of corner cases and grey areas when assessing the compatibility of licenses.

In conclusion, the framework provides great potential for managing IP of a large number of projects and provides very useful interfaces for both developers and IP/license experts. To leverage this potential, respective resources are required.

7 Conclusion

This deliverable provided an overview of the open-source perspective on the TRANSACT project from different angles:

- Eclipse POOSL as a success story of an open source project that started in research, was used and further developed in TRANSACT and commercially adopted.
- The open-source horizontal demonstrator made available in Eclipse Research Labs environment to allow experimentation with a TRANSACT reference implementation in a truly open and accessible way.
- And a view on the topic of open source license compliance which plays an important role in TRANSACT domains that are heavily regulated by safety- and security-requirements but at the same time are already deeply involved with open source developments.

8 Appendix A - Requirements for a continuous compliance tool

ID	1	Priority	High	Type	F
Summary	List licenses and IP in project code and third-party dependencies.				
Description	The tool should provide a human-readable list of the licenses in the project code and third-party dependencies.				
Fit criterion	Human-readable list of licenses is available				
Assessment	YES. ORT provides various reports incl. SBOM and NOTICE, which are human-readable.				

ID	2	Priority	High	Type	F
Summary	Check license compatibility and display those that cannot be automatically validated.				
Description	<p>The tool should be able to check if a license is compatible with Eclipse's rules and display to the end user the ones that can't be automatically validated.</p> <p>Eclipse approved licenses: https://www.eclipse.org/legal/licenses.php#approved</p>				
Fit criterion	Problematic licenses can be shown				
Assessment	YES. Problematic licenses and explanations are provided. PARTIALLY: Sometimes rules for automatic validation were a bit too conservative for our needs and there were too many false positives.				

ID	3	Priority	High	Type	F
Summary	Reuse wisdom from the clearly-defined initiative.				
Description	The tool should be able to query the clearlydefined service to retrieve existing curations and clearance from known, collaborative databases.				
Fit criterion	Clearlydefined data can be used				

Assessment	YES. Clearlydefined data could be leveraged.
------------	--

ID	4	Priority	High	Type	F
Summary	Generate Bill of Materials				
Description	The tool should be able to generate a software bill of materials (SBOM) in SPDX format.				
Fit criterion	SBOM in SPDX is generated				
Assessment	YES. SBOMs are generated in SPDX and CycloneDS formats.				

ID	5	Priority	Medium	Type	NF
Summary	Do not interfere with existing code or build processes: simply clone code and run the analysis.				
Description	We don't want to interfere with projects. Only committers and contributors are allowed to change their repositories.				
Fit criterion	No need to add files to the analysed repository to run the analysis.				
Assessment	YES.				

ID	6	Priority	Medium	Type	NF
Summary	Be technology-agnostic, regarding both the code and the build system.				
Description	There are a huge variety of technologies in the open source projects hosted at the foundation. As an example IOT-related projects often propose reference implementations in various languages to their standardised APIs. Take care of as many build tools as possible.				
Fit criterion	Work with these build systems: Maven, NPM, GoDep, Gradle, Bower, Yarn, Composer, Conan, Python pip, Cargo.				

Assessment	YES
------------	-----

ID	7	Priority	High	Type	NF
Summary	Rely on standards (like SPDX) to identify licenses.				
Description	All Eclipse projects should use SPDX license names, as it has become a established standard. This enables our analysis to easily and accurately detect licenses, and the end users to quickly and safely know what they get.				
Fit criterion	The tool should recognise and use SPDX identifiers in files.				
Assessment	YES				

ID	8	Priority	Medium	Type	NF
Summary	Can be easily executed on any repository (i.e. have a simple script to start the whole process).				
Description	We need to be able to fully automate the analysis process. With hundreds of open source projects (and with many repositories per projects) the analysis cannot be done manually.				
Fit criterion	The tool enables automation and requires no human interaction when executed.				
Assessment	YES. Was tested with regularly scheduled scans.				