Towards Language-Agnostic Reuse of Palladio Quality Analyses

Malte Reimann, Stephan Seifermann, Maximilian Walter, Robert Heinrich
malte.reimann@student.kit.edu, {seifermann,maximilian.walter,heinrich}@kit.edu
Karlsruhe Institute of Technology

Tomáš Bureš, Petr Hnětynka
{bures,hnetynka}@d3s.mff.cuni.cz
Charles University, Czech Republic

Abstract

Palladio is the foundation for many research projects because of its increasing support for various quality properties. However, the Eclipse-based infrastructure does not always integrate well with other tools, which impedes or makes it tedious to reuse existing analyses. The Palladio tooling does not yet provide a language-agnostic interface that would support such integration scenarios. In this paper, we present the architecture and a prototypical implementation of such an interface based on gRPC and REST.

1 Introduction

Palladio [3] is a quality prediction approach for component-based software architectures. Various research projects use its quality analyses or extend them. For instance, CACTOS[1] integrates energy consumption analyses into Palladio. New reliability analyses are the topic of Smartload[2]. KASTEL[3] integrates security and privacy analyses into Palladio. To extend the Eclipse-based Palladio implementation, projects contribute new bundles to the codebase.

Restricting all contributions to be code compatible with the Eclipse ecosystem is not useful. Project partners often have their own tools that also provide a functionality required for the project. For instance, we had to use Palladio from within a complex optimization implementation in our research project Trust 4.0 [4]. Porting Palladio to a non-Eclipse platform is not feasible as well. Therefore, integrating all tools via defined interfaces is the most realistic option. However, Palladio does not provide such an interface.

Existing interfaces to Palladio address performance analyses. *Experiment Automation* [1, 2] provides a command-line interface to performance analyses on the local system. A recently created Docker image [6] encapsulates this interface. Adding further analyses is possible. However, retrieval of results is limited to the local file system. *Simulation as a Service* [4] also uses Docker and adds sophisticated load balancing and scaling strategies. However, none of these approaches provides a generic approach for accessing quality analyses of Palladio and their results from remote systems. Therefore, each research project has to decide about interface technologies and styles.

To ease interface definitions, this paper proposes a reference architecture for interfaces to quality analyses based on gRPC[5] and REST. We assume the performance of gRPC to be better but REST provides superior compatibility. This architecture aims for saving effort and has potential to become a central access point to various quality analyses of Palladio.

We implemented an architecture prototype with an Eclipse-based server and an Eclipse-independent interface to the analyses. To show its feasibility, we integrated a confidentiality analysis. Additionally, we tried to replace gRPC with RMI[6] to investigate the effort in switching communication infrastructures.

2 API Architecture

Our first objective is to use Palladio quality analyses from different programming languages. We need to interact with an interface, without using Palladio or Eclipse directly. To achieve this, we define a gRPC and a REST interface, that hide the underlying Palladio and Eclipse infrastructure. Replacing gRPC with other communication infrastructures like RMI is possible, which we did in a prototype. For a sake of simplicity, we only report on gRPC in the following. The interfaces have to provide functionality to manage projects, upload files, run analyses, and retrieve results of analyses. Secondly, we want to be able to add analyses without having to modify our infrastructure. We allow new services to integrate into the architecture by OSGi service discovery. In the following, we describe the interface and the integration of services.

We propose the server-side architecture shown in Figure 1. The sum of all components shown builds the server. The *gRPC Server* and the *REST interface* provide the interfaces to be used by external

[1]https://cactos.github.io/
[2]http://www.smartload-project.de/
[3]https://www.kastel.kit.edu
[4]http://trust40.ipd.kit.edu/home/
[5]https://grpc.io/
[6]https://docs.oracle.com/javase/tutorial/rmi

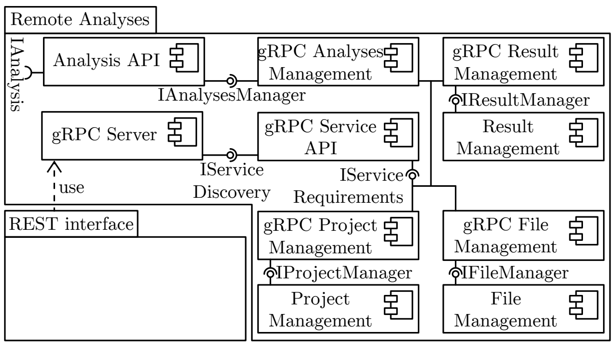

Figure 1: Remote Analysis Architecture (gRPC)

tools. The components of *Remote Analyses* run in an Eclipse application. However, using the interface does not require Eclipse or even Java. External tools employing other technologies can use provided services as clients. gRPC allows our gRPC interface to offer Palladio analyses to thirteen languages[7]. For all other languages, we provide a REST interface. The REST interface realized with Spring Boot translates calls to the gRPC interface. By the REST interface, our architecture becomes language-agnostic. In our architecture, the REST server or a program directly calling the gRPC interface is a client. As a result, each service needs to provide its functionality as gRPC interfaces. There are four predefined services: (1) project management, (2) file management, (3) analyses management, and (4) result management. Services (1) and (2) offer CRUD operations for managing projects and project files. Service (3) accepts input parameters and runs a specific analysis. Service (4) provides the results of an analysis.

A new service has to implement the interface `IServiceRequirements` for two reasons. First, the server automatically discovers and serves the new service via OSGi declarative services, which eases extending existing services. Second, `IServiceRequirements` requires a gRPC interface. Extending the REST backend to use the new gRPC interface exposes the new service. We favor a dedicated business logic component for a service to decouple logic from communication infrastructures. The downside of this approach is that implementors must have a rough understanding of how gRPC works. To ease integrating new analyses, the services analyses and result management provide generic and extensible functionality for all analyses. A new analysis has to implement the `IAnalysis` interface. Again, OSGi declarative services discovers the analysis and provides it to the analyses management. Adding a new analysis does not require changes with the gRPC interface or REST backend. `IAnalysis` requires the analysis to define the input schema it expects. Inputs are specific values or links to files the analysis uses. The analysis must implement an entry point that calls existing analysis logic of Palladio. Every analysis provides a unique identifier (ID) that clients use to invoke the analysis together with required inputs. When invoking an analysis, analyses management passes provided input values to the entry point of the analysis. The server will return immediately with an ID for the analysis run that clients can use to retrieve the results from the result management later. We decide on this behavior because most analyses are long-running, and clients often expect an immediate response. Once complete, the analysis registers the result with the result manager. Any serializable structure for the result works. If an analysis cannot convert its result to a serializable structure, a new service for querying results is a solution.

3 Security Analysis Interface

We demonstrate integrating a specific analysis by a data processing analysis [5] from the context of our research project Fluid Trust. The analysis compares required access rights for transmitted data with roles assigned to components. The analysis needs paths to a usage model, allocation model and characteristics model from the analysis input. Additional inputs are an ID for a logic prover factory, an ID for an analysis goal and optional flags to specify optimizations for the solving process. The data processing analysis integrates into our architecture by implementing the `IAnalysis` interface, as the previous section describes. Our architecture dispatches input from a client to the data processing analysis. It is not possible to describe the full implementation within the page limits, so we refer to our repository[8] for the implementation details.

We demonstrate how to execute an analysis by the ContactSMSManager example available in our Github repository[9]. Executing the data processing analysis consists of the following steps. The clients sends a `POST` request to the `/projects` REST endpoint with a body containing an identifier for a new project. We use *SMS* as the identifier. Second, we create and upload each of the files the data processing analysis needs. We create the `default.repository` file by calling `PUT` on endpoint `/files/project/SMS/default.repository`. Third, we run the data processing analysis on the *SMS* project, with `PUT`, to the endpoint `/launch/dataprocessing/contactSMSManager` and using the JSON body from Listing 1.

Listing 1: JSON Body

```
{
  "usageModelPath":"SMS/default.usagemodel",
  "allocModelPath":"SMS/default.allocation",
  "characteristicsModelPath":"SMS/
      characteristicTypes.xmi",
  "proverFactoryId":"org.prolog4j.tuprolog.
      proverfactory",
```

[7] `https://grpc.io/docs/languages/`

[8] `https://git.io/JUJzr`

[9] `https://git.io/JUfPV`

```
  "analysisGoalId":"pcm.dataprocessing.
      analysis.launcher.returnquery",
  "launchFlags": [],
  "parameters":{
    "Role characteristic type (ID)":"uuid0",
        "Access rights characteritic type (ID
            )":"uuid1"
  }
}
```

The URL contains the ID of the analysis to execute. In this case the ID is *dataprocessing*. The client provides *contactSMSManager* as a meaningful name. Our architecture uses the name to build an ID for the launch. The body contains all inputs required by the data processing analysis. To reference the launch later, the response body returns a ID, for example *contactSMSManager-2020-08-16T10:30:15.329451500Z*. We poll `/results/{ID}` with `GET` to retrieve the analysis results.

4 Discussion

The effort required to integrate a new quality analysis might differ. If it is sufficient to provide inputs and outputs complete at one point in time as described in Section 2, the integration is easy. The analysis only needs to implement one interface. For analysis with more specific requirements, integration is more complicated. If clients cannot provide a complete set of inputs at one time but have to create it assisted by the server, the integration is more complex because dedicated services are required.

So far, there is no tenant support because we did not integrate a concept for Identity & Access Management (IAM) yet. However, gRPC provides a built-in authentication mechanism[10] that we just have to translate to mechanisms compatible with REST. Therefore by using gRPC, the architecture can be extended with IAM. Anyhow, missing IAM is usually no problem in the context of research projects. Since our architecture uses a single Eclipse instance and one workspace, scaling out by using multiple instances of Eclipse and load balancing is not feasible. Storing files in blob storage and not inside Eclipse, where we run analyses, could solve this.

Switching the communication infrastructure between gRPC and RMI was possible but the effort for doing so is considerable. Many infrastructure code has to be replaced and reimplemented. Therefore, we suggest to carefully select the infrastructure and stick to it. So far, we did not experience issues with gRPC. In contrast, RMI is Java-specific. Therefore, many research projects would be excluded from using this interface directly but are forced to use REST.

We considered using REST as the only interface technology but favored a combination of gRPC and REST. Obviously, only using REST would imply less indirection and the performance should be comparable when integrating the REST server into Eclipse.

[10] `https://grpc.io/docs/guides/auth/`

However, we decided to use remote calling between the REST interface and the Eclipse instance for separation of concerns and decoupling. Reusing the REST server architecture for other projects might be of interest. The performance penalty of the REST to gRPC communication is expected to be low when considering that quality analyses usually require much more time than starting the analysis and receiving results.

Different analyses have different result types. This is a drawback when combining analyses into one interface. Right now, we require serializable analysis results. We think this gives enough flexibility. However, streamlining results of all analyses would be beneficial for clients working with the results.

5 Conclusion and Future Work

This paper presented a reference architecture to use Palladio quality analyses in a language-agnostic way.

The primary benefit of the presented architecture is its extensibility with respect to further analyses. This easily allows to implement new specific analysis or provide additional REST endpoints. Additionally, the presented architecture enables the usage of Palladio quality analysis outside the eclipse ecosystem.

We plan to integrate more analyses into the architecture, allowing us to reason on how well the architecture scales. Evaluating the performance of the used technologies gRPC, RMI and REST is also subject to future work. More work on how to streamline Palladio analyses into one interface is necessary. In general, we see an opportunity for new applications of Palladio. For example, offering Palladio as an action for continuous integration pipelines. Building out a front-end based on our architecture will enable us to evaluate the architecture in a real-world scenario.

Acknowledgements This work has been funded by the DFG (German Research Foundation) – project number 432576552, HE8596/1-1 (FluidTrust), supported by the Czech Science Foundation project 20-24814J, and also partially supported by Charles University institutional funding SVV 260451. We thank Sandro Speth (University of Stuttgart) for providing feedback.

References

[1] P. Merkle. "Comparing Process- and Event-oriented Software Performance Simulation". MA thesis. KIT, Germany, 2011.

[2] S. Lehrig. *Quality Analysis Lab (QuAL): Software Design Descriptionand Developer Guide Version 1.1.* University of Paderborn. 2016.

[3] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach.* Cambridge, MA: MIT Press, Oct. 2016. 408 pp.

[4] F. Willnecker et al. "SiaaS: Simulation as a Service". In: *Softwaretechnik-Trends* 36.4 (2016).

[5] S. Seifermann et al. "Data-Driven Software Architecture for Analyzing Confidentiality". In: *ICSA'19.* IEEE, 2019, pp. 1–10.

[6] T. Weber. *Dockerifizierung of Palladio.* Practical Course Report. KIT, Germany. 2020.