2023-09-13

Effective Control System Framework Selection through Checklist-based Software Quality Evaluation

Imani, Alireza

UNIVERSITY OF CALGARY

Effective Control System Framework Selection through Checklist-based Software Quality Evaluation

by

Alireza Imani

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING

CALGARY, ALBERTA

SEPTEMBER, 2023

Abstract

The Herzberg Astronomy and Astrophysics Research Centre (HAA) of the National Research Council (NRC) is Canada's premier center for astronomy and astrophysics. It maintains the largest and most powerful observatories in Canada and represents Canada at many of the world's leading astronomical events. In the context of my pursuit of a master's degree, a collaborative effort unfolded between HAA and myself, centered around the multifaceted project named ARTTA4.

In the realm of control systems, the significance of prioritizing the evaluation of open-source software's quality is undeniable. This emphasis arises from the essential role that a thorough appraisal of these components plays in safeguarding the security, stability, and efficiency of such systems. Neglecting this assessment exposes control systems to a range of vulnerabilities, including bugs and compatibility concerns that could result in operational disruptions, security breaches, and potential risks across diverse industries. Ensuring the integrity and performance of control systems demands a rigorous approach to software quality assessment, serving to preempt unforeseen complications and bolstering the overall reliability and functionality of these systems.

Through my engagement with HAA, I recognized the pivotal role of an open-source control system toolkit named Tango Controls in shaping their antenna control system. Consequently, a comprehensive evaluation of Tango Controls' software quality emerged as a vital undertaking for guaranteeing the ultimate dependability and maintainability of the resultant control system.

Accordingly, we conducted a generalizable checklist-driven software quality assessment

approach to examine Tango Controls. This evaluation brought to light three specific limitations within this open-source toolkit. This finding prompted me to investigate a substitute control system toolkit to replace Tango Control. Thus, we adopted a Component-based Software Development (CBSD) methodology to propose two potential substitute solutions. These alternatives were put into practice through the implementation of a control system module at HAA, in parallel with the utilization of Tango Controls. To quantify their efficacy, we used SonarQube to generate a static source code analysis report. Furthermore, we conducted an empirical comparison centered around the development process spanning all three methodologies. Drawing from empirical and quantitative analyses, it became evident that one of the proposed solutions outperformed Tango Controls in terms of efficacy and performance.

In conclusion, this thesis stands as a pivotal stepping stone in the realm of open-source software selection for the development of industrial control systems. As we move forward, the path to fully realizing the potential of open-source technologies lies in sustained research efforts and collaborative endeavors. By delving into the criteria commonly referenced by industry practitioners, we can glean insights that refine the selection process. Furthermore, the introduction of a natural language processing-based tool holds promise in revolutionizing how we approach open-source software comparison and adoption. Such a tool aims to streamline the process by autonomously aggregating pertinent information from diverse online sources. Through this holistic approach, we aspire to foster an environment where open-source technologies are harnessed to their fullest extent, driving the evolution of industrial control systems and propelling technological advancement.

Preface

Chapter 2 was presented at the *International Conference on Information Systems and Advanced Technologies* (ICISAT 2021) and it is published by IEEE as:

A. Imani, M. Moshirpour and L. Belostotski, "Checklist-based Software Quality Evaluation of Tango Controls," 2021 International Conference on Information Systems and Advanced Technologies (ICISAT), Tebessa, Algeria, 2021, pp. 1-7

Chapter 3 of this thesis was initially submitted to the *International Conference on Computer, Control, and Robotics* (ICCCR 2023), and it received acceptance for presentation. Nevertheless, as we have plans to enhance our analysis and publish it in a different venue in the future, we opted to withdraw our submission from the conference. You can find a screenshot of the notification confirming the acceptance of this portion of our study at ICCCR 2023 in the Appendix.

Acknowledgements

Despite residing thousands of miles distant from my family, I've successfully acclimated to a new university, surmounting fresh challenges through the continuous affection bestowed upon me by my family since I relocated to Calgary. Their unwavering care and support have always inspired me to uphold my scholastic advancements. I wish to convey my profound gratitude to my esteemed mentor, Dr. Mohammad Moshirpour, whose sagacious guidance and unwavering support have been instrumental in shaping my research trajectory and facilitating my transition to the rigors of graduate-level studies.

Equally deserving of my appreciation is my co-supervisor, Dr. Leonid Belostotski, whose constructive feedback and endorsement of my academic choices have been invaluable throughout my master's program. I also wish to express my sincere thanks to the members of Dr. Moshirpour's research lab, particularly Mr. Ali Salmani, whose assistance and encouragement have been of great significance.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
$ALTS$	Application Layer Transport Security
$AMPQ$	Advanced Message Queuing Protocol
API	Application Programming Interface
$COTS$	Commercial of the Shelf
$DRAO$	Dominion Radio Astrophysical Observatory
$ESRF$	European Synchrotron Radiation Facility
GUI	Graphical User Interface
HAA	Herzberg Astronomy and Astrophysics Research Centre
ICE	Internet Communication Engine
JWT	JSON Web Token
NRC	National Research Council
OSS	Open-source Software
RPC	Remote Procedure Call
$SCADA$	Supervisory Control And Data Acquisition
SVE	Shared Variable Engine
TLS	Transport Layer Security

Chapter 1

Introduction

1.1 Motivation

The Herzberg Astronomy and Astrophysics Research Centre (HAA), a distinguished entity within the framework of the National Research Council (NRC), occupies a pivotal role as the foremost hub for astronomy and astrophysics in Canada. It boasts the operation of the Dominion Radio Astrophysical Observatory (DRAO), situated beyond the confines of Penticton in British Columbia. Additionally, the HAA oversees the management of the Canadian Astronomy Data Centre and orchestrates Canadian participation in a spectrum of astronomical endeavors. These encompass the Canada-France-Hawaii Telescope, the Gemini Observatory, the Atacama Large Millimeter Array, the Square Kilometre Array, and the Thirty Meter Telescope. Furthermore, the HAA is responsible for stewarding Canada's national astronomy data center, accentuating its significance as a central pillar within the realm of astronomy and astrophysics endeavors. Therefore, the facilities housed within DRAO are classified as mission-critical infrastructure. Given this criticality, software component quality assurance assumes paramount significance, particularly when these components contribute to the operational functionality of these vital settings.

In the study conducted by Laila et al., a survey involving 110 IT executives was undertaken to discern the factors influencing the adoption decision of Open-source Software (OSS) within the context of Mission-Critical IT Infrastructures [2]. The findings of this research

revealed that within mission-critical settings, the decision to adopt OSS is notably influenced by factors related to OSS quality as well as security concerns.

Tango Controls is a distributed object-oriented control system framework, establishing the groundwork for an underlying communication protocol and an Application Programming Interface (API). As part of my master's degree experiments, I engaged in a collaborative project with the HAA at the Dominion Radio Astrophysical Observatory (DRAO). This undertaking was centered on the development of control system modules. Throughout this collaboration, it became evident that Tango Controls was the primary OSS utilized in developing the control system.

In light of this, conducting a comprehensive evaluation of the software quality embedded within Tango Controls assumed a critical role. This endeavor extended beyond safeguarding the dependability and maintainability of the control system modules under development at DRAO. It also held significance as an academic benchmark, offering a reference point for the broader community of developers engaged with this extensively utilized OSS. The insights gleaned from this assessment are poised to contribute not only to the immediate refinement of the ongoing project but also to serve as an academic reference for addressing any potential shortcomings in subsequent iterations of Tango Controls.

1.2 Research Objectives

The primary aim of this research encompasses the identification of potential threats to the software quality inherent within Tango Controls. Drawing from [3], the landscape of OSS evaluation methodologies is organized into three overarching categories: checklist,

measurement, and hybrid. This delineation ushers in the inaugural research inquiry:

RQ1: What OSS evaluation methodology should be employed to discern the vulnerabilities within Tango Controls' software quality?

Given that measurement-based methodologies furnish quantitative outcomes for assessing software quality, as exemplified by [4] who introduce a metric-oriented evaluation model geared toward open-source software, resulting in categorization into "Excellent," "Good," "Fair," and "Poor," it is important to note that such approaches offer quantitative reflections of software quality across various facets. However, they are limited in their capacity to directly pinpoint threats to software quality within an OSS toolkit. Consequently, we elected to structure our evaluation method around a checklist paradigm. This checklist acts as a repository of evaluation criteria integral to our process.

Acknowledging the generic nature of evaluation methodologies and their inherent applicability across domains, an imperative emerges to tailor these methodologies to the unique landscape of control systems. This adaptation becomes essential to ensure alignment with domain-specific requirements and nuances. Consequently, the ensuing research question takes form:

RQ2: How can the selected evaluation methodology be refined and tailored to harmonize with the distinctive characteristics of the control systems domain?

We embarked on a comprehensive review of academic references, leveraging their insights to compile an evaluation checklist. This compilation was carefully tailored to encompass the array of Supervisory Control And Data Acquisition (SCADA) systems' requirements and the fundamental dimensions crucial to control systems.

After the completion of the evaluation, our focus shifts towards unveiling the drawbacks

associated with Tango Controls. This progression leads us to the subsequent research objective, which centers on the identification of an appropriate substitute for Tango Controls. To achieve this objective, the initial step is to curate a roster of potential candidate replacements that effectively address the vulnerabilities of Tango Controls. In this pursuit, our goal is to ensure that these candidates remain unaffected by Tango Control shortcomings. This leads us to the following research question:

RQ3: How can we effectively identify potential candidates to replace Tango Controls, while guaranteeing their immunity to the recognized limitations?

Recent growth in open-source software use has spurred the adoption of a component-based software development (CBSD) approach, which aims to avoid unnecessary redundancy. This methodology uses existing software components to construct new software products, thus eliminating the need to reinvent existing solutions. Hence, each potential candidate is crafted by integrating existing software components. This approach not only expedites the development process but also taps into the cumulative expertise embedded within these components, enhancing the efficiency and effectiveness of the resulting candidates.

After compiling a list of potential substitutes, the next step is to choose the most qualified candidate. By examining the features and qualities of these candidates, we can assess their capability to address the identified issues. To aid in this evaluation, we require a method to compare candidates against Tango Controls, focusing on the quality of the underlying code. This evaluation process is pivotal in the pursuit of optimal software solutions. Serving as a critical link, this process guides us toward a thoughtful decision that not only tackles the current shortcomings but also steers us toward improved software quality. Consequently, the final objective of this thesis is framed as:

RQ4: How can we rigorously evaluate and select the most suitable candidate among replacements and Tango Controls?

Our analysis of the candidate solutions we put forth, in comparison to Tango Controls, unfolds through the practical implementation of a control system component at DRAO called Antenna Controller. This practical examination involves employing the candidate solutions to construct this component, thereby subjecting them to a real-world test within the DRAO environment. The resulting source code from each implementation is then compared quantitatively using static source code analysis and qualitatively through empirical analysis.

1.3 Research Contributions

The contributions of this thesis can be summarized as follows:

1. Investigating the vulnerabilities of Tango Controls, a widely used control system toolkit, and recommending an alternative.

2. Introducing a comprehensive checklist for the evaluation of control system software toolkits. The methodology employed in the creation of this checklist can seamlessly be replicated in alternate contexts, enabling the construction of tailored assessment criteria.

3. As an academic reference, our methodology serves as a guide to dissecting potential threats to OSS libraries' software quality through the lens of their documentation.

4. Providing an example of comparing OSS libraries by implementing a case study

and conducting quantitative and qualitative analyses

1.4 Significance of the research and thesis organization

The analysis conducted to uncover threats to the software quality of Tango Controls has led to the identification of its inherent drawbacks. This revelation holds substantial value for the TANGO Collaboration Steering Committee, as it equips them to adjust their roadmap in light of these insights. Furthermore, our published findings [5] serves as a cautionary measure for prominent industrial users of the toolkit, particularly those situated in pivotal industrial contexts. The publication effectively highlights potential weaknesses within the toolkit, which prompts a proactive response from users. This assertion gains credence as indicated by the engagement of key figures from industrial entities. Notably, the head of the Control System Section at ALBA and the head of the Software group at ESRF initiated discussions based on the published paper. This direct interaction underscores the significance of the research findings in influencing and guiding industrial practices and decisions within the realm of control system toolkits.

Moreover, this research marks a significant stride toward domain-specific software quality assessment. Considering the unique characteristics of each domain, tailoring software quality evaluation methodologies is crucial. This study serves as a high-level pipeline that can be implemented by various mission-critical industrial and governmental settings to continuously test the software quality of the OSS libraries they are using and to identify potential alternatives if necessary.

The remainder of this thesis is organized as follows:

In Chapter 2, we first discuss our methodology for selecting an evaluation approach to assess Tango Controls' software quality. Subsequently, we unveil the intricacies surrounding the customization of this evaluation method, aligning it with our unique requirements. The chapter then progresses to a detailed exposition of two overarching categories within the control systems domain, thereby broadening our purview to include domain-specific considerations. Lastly, we shed light on the existing mechanisms inherent in Tango Controls that pose potential risks to our customized software quality criteria. This chapter answers RQs 1 and 2.

Chapter 3 elaborates on our methodology to compile a list of candidate replacements for Tango Controls. We thoroughly discuss each candidate's capabilities and their ability to cover Tango Controls' drawbacks. Following this, we transition to the practical application of these candidates through a case study conducted at DRAO. This practical phase involves the implementation of the candidates and a subsequent evaluation using SonarQube-generated static source code analysis. The chapter then shifts focus to a developer-oriented perspective, engaging in an empirical analysis of the candidates. This perspective allows us to discern and discuss the unique strengths that each alternative presents compared to others. The superior alternative to Tango Controls is introduced to fulfill RQs 3 and 4.

In the end, chapter 4 summarizes the thesis, discusses the limitations of this study, and explores future directions of the study to facilitate OSS evaluation and adoption in real-world applications by employing natural language processing.

Chapter 2

Checklist-based Software Quality Evaluation of Tango Controls

2.1 Abstract

Tango Controls is an open-source framework for distributed control systems used by a growing number of industrial and institutional partners. Despite the many benefits it provides to users, such as a growing community, industrial support, highly scalable, and so on, there are some disadvantages to this trending framework. Uncovering these drawbacks creates users' awareness while considering using this open-source software in their control systems and motivates the Tango Controls development community to fix these issues.

In this chapter, first, we review the research conducted to evaluate, optimize and compare Tango Controls with other trending commercial and open-source control system frameworks. Afterwards, we evaluate Tango Controls via a checklist-based approach by considering two types of control systems and scrutinizing the frameworks' documentation. As a result, we detect reliability and security drawbacks that have not been identified in relevant studies as the first step of our future work to optimize Tango Controls by introducing a toolset.

2.2 Introduction

A Control System (CS) is a manual or automated mechanism employed to handle real-time processes by maintaining or setting physical quantities such as mass, temperature, or speed [6]. Supervisory Control And Data Acquisition (SCADA) is a software package acting as an interface to hardware modules [7]. SCADA is widely used in industry to control varying facilities like controlling a whole oil refinery [8], transport of oil, gas, water, electrical power grids, and railway systems [6]. Like many other softwares, there are Commercial Off the Shelf (take FTV-SE and PVSS as examples) and open-source (Tango Controls and EPICS, for instance) SCADA packages suitable for diverse control systems, making it challenging to decide on which to opt for [9]. Thus, research has been done to evaluate and compare different SCADA packages, facilitating future users' decisions.

Tango Controls was initially proposed in a paper in 1998 by W-D. Klotz, A. Götz, E. Taurel, and J. Meyer. It has been created based on an RPC-based control system called TACO. The Tango Controls key concept, devices in a device server, was an improved version of what had been developed in TACO. Tango development was initiated in 1999 at the ESRF (European Synchrotron Radiation Facility), and the latest major release was made in September 2015 [10]. Over 40 international partners and institutions use this tool kit worldwide, reflecting that Tango Controls is well-maintained, highly scalable, easy to use, and reliable [11].

Although Tango Control is a mature control system framework, since it is an open-source candidate for industrial parties that plan to choose a SCADA package, evaluating it against control system requirements and challenges, not only helps the industry as an academic

reference but also provides the Tango Controls community with potential long-term goals to enhance their framework. Thus, we performed this research to answer two research questions:

- RQ1: What are the criteria that should be considered during the evaluation process?
- RQ2: How can we evaluate Tango Controls concerning open-source software characteristics?

Hence, in section 2.3, we review the literature for any study relative to evaluation, optimization, and comparing Tango Controls to other well-known control system frameworks. Moving forward to section 2.4, we elaborate on our evaluation methodology and evaluate Tango Controls using a checklist-based from new aspects that have not been addressed by related works. We conclude the results and contributions of this chapter in section 2.5, and finally, we introduce our future work to address the identified drawbacks in section 2.6.

2.3 Literature Review

We have reviewed studies relevant to our research, which includes any recent research conducted on optimization, evaluation, and comparing Tango Controls in different terms such as GUI consistency, accessibility, interoperability, ease of application setup and performance. Moreover, since Tango Controls is an open-source software, we have studied a recent systematic literature review on open-source software evaluation, selection, and adoption methodologies. To assess the reliability of Tango Controls, we also review a recent novel component-based software reliability assessment method in this section. We elaborate on each of the studies as mentioned earlier next.

GUI consistency matters may apply to any cooperative project on which members need to adjust the generic tools to their contexts. Tango Controls consists of various services and toolkits managed by a group of applications developed either by the core team or other community members [12]. ALBA is a third-generation Synchrotron Light facility in Barcelona, the latest source in the Mediterranean region [13]. At ALBA, the GUI inconsistency problems were addressed by deploying Taurus as the default GUI framework for all their applications [12]. Taurus is a python framework for data acquisitio and control interfaces that supports a variety of control systems or data sources such as Tango Controls and EPICS [14]. Manrique et al. [12] have implemented a Taurus-based application to unify Tango Controls services in a customizable graphical user interface.

In terms of accessibility, Goryl et al. [15] have elaborated on projects and activities performed by the Tango Community to promote collaboration and make starting with Tango Controls smoother. Briefly, the projects are as below:

- Unifying Tango Controls documentation from TangoBook, a pdf file provided with the source, and many other documents in a collaborative way that enables the community members to suggest enhancements and contribute to the new Tango Controls Documentation.

- Developing and maintaining a Tango Controls demo virtual machine image, namely TangoBox, to provide:

  - A way to use Tango Controls without putting effort into the investigation, selection and configuration of most of Tango Controls

  - A working configuration example
  - An environment for developers to create and test new software

- A web application called *Device Classes Catalogue* [16] to consider effective search if a particular device is already supported. It encourages open-source software, allows for software reuse, and hinders rework.

Bourtembourg et al. [17] have conducted a research on mitigating the limits of HDB++, Tango Controls archiving system, using PostgreSQL and Timeseries databases. They have compared the different database backends supported by HDB++, e.g. MySQL/MariaDB, Cassandra, PostgreSQL, TimescaleDB, and Elasticsearch. Using a Github repository containing HDB++ archiving system benchmarking tools, they have evaluated the performance of the supported database backends in different scenarios such as insertion, scalar attribute query, and arrays query benchmark. It has been concluded that the PostgreSQL and TimescaleDB backends outperforms when dealing with Tango spectrum attributes.

Furthermore, Drochner et al. [18] have reported the problems observed while switching from TACO to Tango Controls in a neutron scattering instrument control system. After facing "serialization timeout" errors, they realized that in a Tango Controls device server, no command can take more than 3 seconds to be executed. While trying to add individual "busy" states explaining to all clients what a server is currently waiting for, it has been observed that using custom values in Tango Controls states would cause confusion in other parts of the framework.

Bolkhovityanov and Cheblakov [19] have performed a comparative analysis of the architecture of control systems of physical research facilities. They have compared Tango

Controls, EPICS, and CXv4 in terms of interoperability. According to their research, in a Tango Controls system, the client-server exchange is achieved within the paradigm of devices, rather than through channels, limiting access to other control systems for Tango Controls users. Below summerizes their analysis and the obtained results:

- Tango Controls does not provide direct access to other control systems.
- EPICS supposes that control systems operate only under EPICS, but allows the creation of gateway servers to other control systems.
- CX allows the direct data exchange from an arbitrary control system.

Research has been conducted to choose a SCADA package for a new device at Consorzio RFX by Barana et al. [9]. They performed tests on two commercial (FTV-SE by Rockwell Automation and PVSS II by ETM) and two open-source packages (EPICS and Tango Controls) SCADA packages to evaluate them in ease of application setup and performance. According to their results, despite Tango Controls being better than other candidates in terms of performance, since understanding the Tango Controls structure required more effort than the other SCADAs, substantial programming expertise was required, and a remarkable effort was unavoidable to implement the data exchange in an effective way, they discarded Tango Controls as an option to be used to develop the new device.

A systematic literature review has been performed by Lenadruzzi et al. [3] on studies concerning open source software evaluation (OSS), selection, and adoption methodologies, factors, measures, and information that characterize the analyzed models. Having reviewed 60 studies concerning open-source software evaluation, selection, and adoption, they have

made a comparison between the studies in terms of the importance of different measures and factors involving in OSS selection, evaluation, and adoption. They have classified the studies into three categories based on what the studies are based on: Checklist, Measurement, and Hybrid. We discuss these categories in section 1.4.

Moreover, Barcelos et al. [20] have conducted a systematic literature review to identify existing architectural evaluation approaches. They have reviewed 20 studies proposing an architectural evaluation approach through December 2004. Similar to [3], they have grouped the studies based on the evaluation techniques and have classified them into questioning, measuring, and hybrid techniques. Besides, they have proposed a checklist-based inspection approach architectural evaluation method to cover domain-specificness and solution-specificness of other checklist-based methodologies.

In a recent study by Chen and Yan [21], a novel reliability assessment method by taking into account the effects of components has been proposed. They have proved the effectiveness of their method by evaluating three existing examples. Their paper has defined component importance for each component in component-based software based on three factors: self-influence, failure influence, and propagation influence. Briefly, if we consider a component C_i, its self-influence depends on the number of visits to it during a single complete software system operation. Also, its failure influence depends on the number of components information flows from them to C_i. Finally, the propagation influence of C_i relies on the count of components to which information flows from C_i. Thereby, the importance of component C_i is calculated by the weighted sum of the three mentioned factors for it. Then, the reliability of a software system can be calculated by multiplication of r_i^{α} for each component C_i, where r_i indicates the reliability of C_i and α is the calculated component importance for C_i.

2.4 Tango Controls Evaluation

As briefly mentioned in section 1.3, according to [3], there are three general categories of open-source software evaluation methodologies depending on what they are based on: checklist, measurement, and hybrid. Since measurement-based methodologies generate quantitative results to evaluate software quality (as an example, [4] have presented a metric-oriented software quality evaluation model, specifically targeted to open-source software, and categorized the result of evaluating software quality aspects into four categories: Excellent, Good, Fair, and Poor), they only reflect the software quality from different aspects and cannot be used to directly identify the threats to software quality in an open-source software product.

Accordingly, we opted for basing our evaluation method on a checklist. The checklist should help as a collection of evaluation criteria in our process. Thus, we reviewed academic references to compile the checklist with a set of supervisory control and data acquisition (SCADA) systems' requirements and essential aspects of control systems.

Alcaraz et al. [22] have identified five control system requirements that are real-time performance, sustainability, dependability, survivability, and safety critical. Industrial control systems requirements have been discussed by Stouffer et al. [23]. They have mentioned data confidentiality and integrity, timeliness, availability, continuity, pre-deployment testing, and resource-constrained. More specific research on non-functional requirements in distributed control systems has been conducted by Frank et al. [24]. Installability, modularity, reusability, analysability, testability, interoperability, time behaviour, resource utilization, reliability, fault tolerance, performance efficiency, compatibility, maintainability, and portability are the critical non-functional requirements mentioned in the study.

Table 2.1 presents the checklist we compiled from the abovementioned studies after removing identical and not applicable requirements. For instance, according to [25], Dependability is a combination of Availability, Reliability, Safety, Confidentiality, and Integrity. Another example is Sustainability. Since the sustainability of a software system depends on the (hardware) resources used [26], it does not apply to our evaluation.

To evaluate our checklist items, we have considered the characteristics of two types of control systems: general control systems and distributed control systems. Next, we will elaborate on each of the types and evaluate the relevant principles of Tango Controls to the traits of that specific type by referring to the Tango Controls documentation publicly available on the readthedocs platform [14]. Our goal was to detect any subsystem, functionality, or mechanism that violates our checklist items according to their criteria and definitions during our crawling process. The next subsections discuss the results of our evaluation.

2.4.1 General Control Systems

Every control system constitutes a set of hardware devices. SCADA acts as an interface to control, monitor, and manage these devices, as stated in the introduction. Thus, the logic behind defining and characterizing each hardware device in SCADA reflects on the general performance of a control system.

Correspondingly, Tango Controls has been built around two concepts called devices and device classes [27]. Devices are objects that have the ability to establish access to their pipes, properties, attributes, and commands as defined by their device classes [28]. Device servers are processes implementing a set of device classes. Device classes translate hardware communication protocols to Tango Controls communication [27].

Each Tango Controls system is associated with a centralized database called Tango Host. Tango Host stores configuration data used at startup of device servers, serves as a name server for dynamic network addresses, and acts as a repository for retaining settings that need to be kept. An extensive system may consist of tens of thousands of devices [27]. All these devices depend on Tango Host to continue functioning correctly.

Table 2.1: Our Checklist Items With Their Corresponding Criteria or Definitions

Checklist Item	**Criteria [22, 23, 29–31]**	**Definition [32–34]**
Real-time performance	Fault tolerance, Fault forecasting, fault detection, fault prevention, fault removal, maintainability, and coordination	
Survivability		A system's ability to provide a pre-defined minimum level of service if it is threatened by one or more specified threats.
Safety-Critical		When the failure of a system could result in unacceptable consequences, then it is safety-critical.

Table 2.1: Our Checklist Items With Their Corresponding Criteria or Definitions

Checklist Item	**Criteria [22, 23, 29–31]**	**Definition [32–34]**
Data confidentiality and integrity		How well the software product protects data and information from unauthorized access, either accidentally or consciously. Ascertaining the completeness and accuracy of assets.
Timeliness		When performing its function, under stated conditions, how responsive and fast the software product is with its response time, processing speed, and throughput rate.
Availability		A software component's availability and operational state at the time of request.
Continuity	Avoiding unexpected outages, easily get stopped and started without affecting production, redundant components	

Table 2.1: Our Checklist Items With Their Corresponding Criteria or Definitions

Checklist Item	**Criteria [22, 23, 29–31]**	**Definition [32–34]**
Resource Utilization		Under stated conditions, the amount and types of resources the software product uses when it performs its function.
Fault Tolerance		When a software product is subject to software faults or infringements of its specified interface, the extent to which the product can maintain a specified level of performance.
Analysability		An assessment of how well the software product may be diagnosed for its deficiencies or causes of malfunctions, as well as identified which parts of the software may need to be modified.
Modularity		Composition of a system or computer program as such that a change to one component will not adversely affect the others.

Table 2.1: Our Checklist Items With Their Corresponding Criteria or Definitions

Checklist Item	**Criteria [22, 23, 29–31]**	**Definition [32–34]**
Interoperability	Communications commonality, data communality	Specific characteristics of software that determine its ability to interact with specific systems.
Installability		A software product's ability to be successfully installed and uninstalled within a designated environment.
Portability	Complexity, concision, consistency, expandability, generality, modularity, self-documentation, simplicity Self contentedness, device independence	Transferability of systems or components between environments (Extension of hardware or software).
Maintainability	Concision, consistency, modularity, instrumentation, self-documentation, software independence	A software product's ability to be modified. Corrections, improvements, or adaptation of the software to the environment, as well as requirements and functional requirements may be part of modifications.

Table 2.1: Our Checklist Items With Their Corresponding Criteria or Definitions

Checklist Item	**Criteria [22, 23, 29–31]**	**Definition [32–34]**
Compatbility		The ability of multiple software components to communicate with one another or to perform their functions while they share a common hardware or software environment.
Reliability	Accuracy, complexity, consistency, error tolerance, modularity, simplicity Self contentedness, accuracy, completeness, robustness/ integrity, consistency Frequency and security of failure, Recoverability, predictability, accuracy, mean time between failure	When used under specified conditions, the extent to which the software product can maintain a specified level of performance.
Reusability	Generality, hardware independence, modularity, self-documentation, software independence	A software asset's potential for reuse in the development of other assets, as well as in multiple software systems.

Table 2.1: Our Checklist Items With Their Corresponding Criteria or Definitions

Checklist Item	**Criteria [22, 23, 29–31]**	**Definition [32–34]**
Testability	Audit ability, complexity, instrumentation, modularity, self-documentation, simplicity Accountability, communicativeness, self descriptiveness, structuredness	To what extent can modified software be validated by the software product.

According to the explained reliability assessment method in [21], considering a Tango Controls system, if we think of each device as a component in a component-based software, information flow from every component to Tango Host and vice versa. Therefore, the impact of the reliability of the Tango Host component on the control system reliability will be significant since α is a relevantly large number due to the high importance of the Tango Host component when computing r_i^{α}. Thus, the existence of Tango Host can undermine a Tango Controls system's reliability.

2.4.2 Distributed Control Systems

Large-scale distributed control systems (DCS) often are consisted of dozens of interacting units. Automated highway systems, airplane formation flight, and satellite constellations are examples of such control systems [35]. Due to the variety of human resources skills, available facilities, and other considerations, each unit might have been developed using a different software control system framework/toolkit, e.g. unit A uses Tango Controls and unit B with

which unit A interacts uses EPICS. In such cases, the interoperability of units should be assured.

To enable integration with third-party technologies, Tango provides REST API specifications. REST API requests are processed through HAProxy configured to use HTTPS protocol for secure communication. Tango REST server that HAProxy interfaces with provides access to a single Tango Host in which a device of class ForwardComposer is defined. This device gives read-only access to the MStatus Tango device with status information about the storage ring at ESRF. Finally, Tango REST API uses Tango Access Control to validate each request [1]. Fig. 2.1 shows the process as mentioned earlier for each REST API request. The following discusses two elements of the elaborated process by referring to Tango Controls documentation and literature.

Tango Access Control There are two tables in the centralized Tango database of a control system that store all the user rights. A device server called TangoAccessControl accesses these tables directly, and only one device can configure it. However, although having a controlled access system running, it is possible to circumvent it by setting the environment variable *SUPER_TANGO* to *True* in the client's application environment [36].

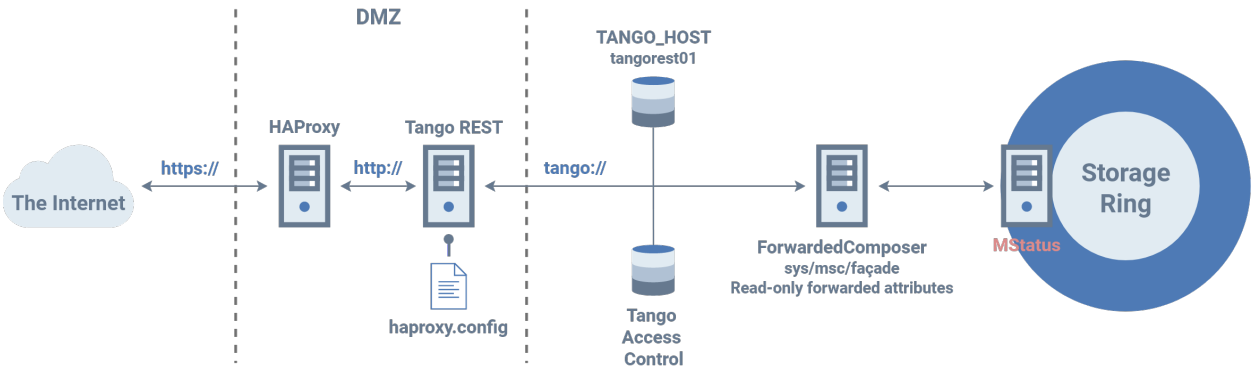

Figure 2.1: Tango REST API Request processing procedure [1]

HAProxy A rapid I/O layer, HAProxy is non-blocking and event-driven with a multi-threaded scheduler based on priorities. By assigning connections to the same CPU as much time as possible, this optimizes the CPU cache efficiency [37]. Nadig [38] has investigated different available service proxies in his thesis. Concerning HAProxy, he has stated that that:

- It does not have a pluggable architecture.
- It does not integrate with a remote service discovery service.
- It does not integrate with a global rate limiting service.

2.5 Conclusion

We reviewed studies relevant to the evaluation, adoption, and optimization of Tango Controls. Although they have reported issues regarding the high programming skills that are required while using Tango Controls to develop a control system, interoperability issues, application setup, and performance, none has stated our detected drawbacks. In addition, we discussed two systematic literature reviews concerning open-source software evaluation and adoption and software architecture evaluation approaches to opt for a suitable evaluation method.

Having explained our methodology, we evaluated Tango Controls using a checklist in two common types of control systems, which lead us to identify deficiencies with Tango Controls that violates some of our checklist items according to their criteria and definitions. Table 2.2 summarizes our evaluation results.

Table 2.2: Violated Checklist Items

Drawback Description	**Violated Checklist Items**
The existence of a centralized database with vital stored data	reliability, availability, continuity
The possibility of bypassing TangoAccessControl	data confidentiality and integrity
The existence of HAProxy in REST API request processing procedure	interoperability

Accordingly, by elaborating our evaluation approach and arguing the results, this chapter contributes in two ways:

- Creating a checklist of software control systems' evaluation criteria and aspects
- Evaluating Tango Controls using the checklist and reporting the identified reliability and security issues

In future, evaluation of other requirements that cannot be detected by documentation analysis like maintainability, testability and other metric-based control systems' requirements can be considered. Furthermore, we plan to devise a solution pack of open-source tools to address the detected deficiencies of Tango Controls. We will evaluate our proposed toolset by implementing a real-world astronomical case study using Tango Controls and our suggested solution pack.

Chapter 3

Towards Replacing Tango Controls: A Comparative Empirical Study

3.1 Abstract

In previous chapter, we evaluated Tango Controls software quality using a checklist-based approach. Accordingly, we identified drawbacks threatening reliability, availability, continuity, data confidentiality and integrity, and interoperability. Our purpose in this chapter is to remediate the deficiencies of Tango Controls by describing two scenarios for deploying multiple open source projects/frameworks to form a control system middleware framework. We have examined the two scenarios in the context of a real-world astronomical project at the Dominion Radio Astrophysical Observatory (DRAO). After conducting the static code analysis and an empirical analysis, we compared and analyzed the project developed using the two scenarios and Tango Controls in terms of the maintainability and complexity of the software. In conclusion, we introduced the scenario that achieved superior results in comparison and discussed its advantages over Tango Controls.

3.2 Introduction

As a distributed object-oriented control system framework, Tango Controls defines an underlying communication protocol and an Application Programming Interface (API) and

provides a set of tools and libraries to help programmers build software for industrial control systems, particularly Supervisory Control and Data Acquisition (SCADA) systems [27]. In previous chapter, we aimed at evaluating the software quality of Tango Controls. There are multiple studies with a similar objective, but they have generally aimed to compare Tango Controls with other open-source and Commercial Off-the-Shelf (COTS) alternatives. Our approach has been focused on identifying structures and procedures that compromise software quality. For the purpose of evaluation, we compiled a checklist of control systems requirements from literature and evaluated Tango Controls in different typical scenarios of control systems such as general and distributed control systems. Accordingly, we identified some drawbacks that threatened our checklist items. The results are available in Table 2.2.

Frameworks and libraries employed in software development play a major role in determining the overall quality of the final product. Using a library migration in Java, Alrubaye et al. investigated the impact of library migrations on software quality and code readability [39]. As a result of their analysis, software quality attributes such as coupling, cohesion, and cyclomatic complexity can be optimized through library migration. Additionally, readability of the code can be optimized for reasons such as more meaningful method names to achieve the same functionality in the new library.

The Component-based Software Engineering (CBSE) technique refers to a methodology for building software using pre-made, reusable software components [40]. According to a recent study by Chatzipetrou et al. [41], components can be selected from the following four categories: (1) Open-source Software (OSS), (2) Commercial of the Shelf (COTS) Software, (3) Internally Developed Software, and (4) Outsourced Developed Software. Based on the results of their research, open-source software has been the practitioners' second popular

choice as a component selection option.

In this chapter, we propose replacing Tango Controls with a solution pack of open-source software not only to address the determined shortcomings, i.e., third-party integration, data storage and exchange, and assurance of the security of the control system, but also to enhance the software quality attributes in a control system. Our solution pack has been proposed to use open-source software to enable the final resulting control system to employ the CBSE approach and to facilitate component selection for the developer of such a control system.

The rest of this chapter is organized as follows: we discuss recent related works in open-source software adoption, selection, framework development, and middleware framework trends in Section 3.2. Section 3.3 presents two candidate solutions resulting from the combination of cutting-edge open source projects. To evaluate the efficiency of our candidate solutions, we take advantage of a real-world control system as a case study and implement the devices using our potential solutions in Section 3.4. We explain our methodology for evaluating candidate solutions in Section 3.5. We present in section 3.6 a quantitative analysis of the case study implementations using the possible alternatives to Tango Controls acquired through static code analysis. Our analysis in Section 3.7 examined each implementation empirically and discussed the differences between them using various terms. The potential threats to the validity of the methodology of this chapter are presented in section 3.8. In our final section, Section 3.9, we will conclude the chapter.

3.3 Literature Review

In order to collect a set of candidate open-source solutions to address each identified deficiency, first we need to review literature for studies concerning component adoption criteria, and risks especially those related to open-source specific materials.

Morandini, Siena, and Susi have conducted a systematic literature review on the risks within OSS versus COTS components adoption in order to derive appropriate measures [42]. They have used the collected knowledge from literature along with the measures available on OSS projects websites to build evidence graphs. Such graphs are being used in their proposed decision making procedure to assess and rank the risks of OSS projects.

Having employed cumulative voting, Chatzipetrou et al. [41] have conducted an anonymous survey among industry practitioners responsible for component selection to understand the characteristics of components that are most important to consider when selecting new components. According to their results, cost, longevity prediction, and support of the component have been the most important attributes that are being considered by practitioners.

Similarly, Butler et al. [43] have anonymously surveyed 13 individuals from six Swedish software companies to analyze contemporary approaches to OSS component adoption and the challenges that are faced during this procedure at the attending companies. According to the responses of interviewees, the authors have categorized the challenges into four groups as follows: Technical aspects, License related matters, OSS project attributes, and Risks. For each class, items have been gathered from responses and discussed in detail.

Spinellis has explained how to choose open source components by introducing 13 criteria [44]. This work has separated the considerations into product and process related

examination. Several examples of questions have been offered for each criterion, as well as hints or guidelines on how to assess the criterion.

A framework for component-based software development has been introduced by Khan et al. [45]. They have proposed a component selection approach by considering standards, component functionality, and component cost. They have also mentioned "replacement" as part of their framework. That is, the component should be substituted due to changes in requirements or emergence of new functionalities.

Al-Debagy et al. investigated the performance of microservices compared with monolithic architectures based on throughput, response time, and number of fulfilled requests [46]. Based on their result, monolithic architecture outperforms microservices with a low number of users. Moreover, monolithic applications are capable of handling requests more rapidly.

A discussion of the design of the distributed control system implemented at the Iranian National Observatory telescope (INO340) has been conducted by Ravanmehr and Jafarzadeh [47]. As part of their implementation of a three-tier hierarchical architecture, they have opted for a publish and subscribe model and Shared Variable Engine (SVE) as the middleware framework. The criteria for selection have been cited as being widely used, and supporting multiple operating systems and programming languages.

Similarly, Kirill has described a software architecture that is capable of reducing development time, allowing an efficient means of exchanging data, and controlling mobile robots [48]. For this architecture, they have considered supporting multiple platforms, having a minimal size of software components, and having low prior knowledge of requirements. ZeroMQ messaging library has been employed for the purpose of command transmission between mechatronic devices to address multi-platformity criterion. In addition, they have suggested

using JSON format to communicate control commands and data.

In a study relevant to our case study, Li et al. have discussed trends in architecture and middleware of radio telescope control systems [49]. They have categorized the software architectures used in radio telescope control systems into three different classes based on time periods, i.e. before 1990's, between 1990 and 2000, and after 2000. They have mentioned Ice, Tango Controls, and EPICS as the trending middleware frameworks as replacements of ACS and CORBA.

García-Valls, Garrido, and Díaz have conducted a research on comparing a set of selected middleware frameworks based on their run-time architecture types [50]. They have categorized middleware frameworks from this perspective into two groups: direct execution, and gateway execution. Therefore, they selected Ice C++ and Corba as representatives for direct execution and Ice C# and Advanced Message Queuing Protocol (AMPQ) as examples of gateway execution architectures to be compared in terms of the amount of time needed by both architectures for executing a predefined remote operation with a fixed processing time. According to their results, Ice C++ outperforms all other candidate middleware frameworks by having the lowest overhead time in a lower-performance hardware environment. Built by ZeroC company, Ice (Internet Communication Engine) is a middleware framework that provides benefits such as being object-oriented, easy to learn and use, and having efficient resource usage, and a built-in security. In addition to the mentioned advantages, Ice supports multiple programming languages, operating systems, and OS architectures [51].

3.4 Available Candidate Solutions

Considering the growth of open source software in recent years, to prevent reinventing the wheel, we have used a Component based software development (CBSD) approach. In this method, a software product is developed by integrating software components to form each candidate. According to Synopsys, the average number of open source components employed in a software product raised by 259% from 2015 to 2020 [52]. Because Tango Controls, the framework we evaluated, is also open-source, we decided to implement our proposed toolkit by selecting open source components for each required functionality. In this section, we will review each Tango Controls deficiency separately and discuss potential open source components that have the capability to address that shortage through the functionalities they provide.

The first drawback is the existence of a centralized database with vital stored data. Instead, we suggest using an alternative communication protocol for the devices in a control system to keep connected to each other and continue collaborating. According to the conducted literature review, we will review the candidate solutions to replace Tango Controls.

3.4.1 gRPC + Envoy

gRPC is an open-source, supercharged Remote Procedure Call (RPC) framework that can be used in any environment. It can efficiently connect polyglot services in a microservice-style architecture [53]. Two gRPC features that encouraged us to consider deploying it in our framework are: Diverse supported authentication mechanisms and gRPC server reflection.

First, we need to define a control system schema in gRPC. We think of control system

devices as a set of highly encapsulated software components that perform well-defined tasks. Each device defines a set of services specifying the methods that can be called remotely by other devices. It also has a stub (client) that provides the same methods as it requires to call from other devices (servers) [54]. In this way, each device in a control system can be comprised of gRPC servers and clients.

Having identified how devices are defined, we may begin the process of decentralizing Tango Controls by repurposing one of the Tango Host roles that is acting as a name server by storing dynamic network addresses. Burns et al. [55] have conducted research on design patterns for container-based distributed systems. They have mentioned the Sidecar pattern as the first and mostly adopted pattern for multi-container deployments, providing reusability, and preventing failure propagation. The sidecar proxy pattern handles communications among microservices [56]. [57] has mentioned Linkerd [58] and Envoy [59] as two popular sidecar proxies over the last years. Among the primary reasons given by the authors to choose Envoy over Nginx is the ease of implementing a fully functional traffic management system. Furthermore, according to Nadig's [38] research, Linkerd requires substantially higher CPU and memory requirements. Contrary to Envoy, Linkerd provides a basic configuration language and does not support hot reloads, but instead it relies on dynamic provisioning and service abstractions. Thus, We opt for Envoy to enable service discovery in our framework. Envoy is an L7 proxy and communication bus designed for large modern service-oriented architectures. It has been created based on the belief that the network should be transparent to applications. When network and application problems do occur, it should be easy to determine the source of the problem [60].

One of the high-level features of Envoy is gRPC support. It supports all of the HTTP/2

features needed to be used as the routing and load balancing substrate for gRPC requests and responses [60]. According to Envoy's terminology, endpoints are network nodes grouped in clusters and implement a logical service. Endpoints in a cluster are upstream of an Envoy proxy [61]. Envoy originated as a service mesh sidecar proxy and removes the responsibility of load balancing, routing, observability, security, and discovery services from devices. In the service mesh model, requests flow through Envoys as a gateway to the network. Each Envoy is equipped with A. Ingress listeners receive requests from other nodes and forward them to the local application. Envoy flows back responses from the local application to the downstream. At the same time, egress listeners take requests from the local applications and forward them to other nodes in the network [61]. Accordingly, each device in a control system that uses this candidate solution is counted as a node in a service mesh and is associated with an Envoy.

On the other hand, each device is responsible for storing its startup and any other configuration data in a local database accessible only by itself. Hence, we could remove the role of Tango Host without leaving any of its primary tasks undone. Figure 3.1 illustrates a device in a control system using our framework.

To ensure the security of our framework, we can leverage the many features offered by gRPC. One of these features is the authentication mechanisms that gRPC provides, including SSL/TLS, ALTS (Application Layer Transport Security), and Token-based authentication with Google. gRPC allows users to extend gRPC to plug in their customized authentication mechanisms [62]. Moreover, Envoy supports multiple security protocols such as TLS, JSON Web Token (JWT) Authentication, External Authorization, and Role-Based Access Control [63].

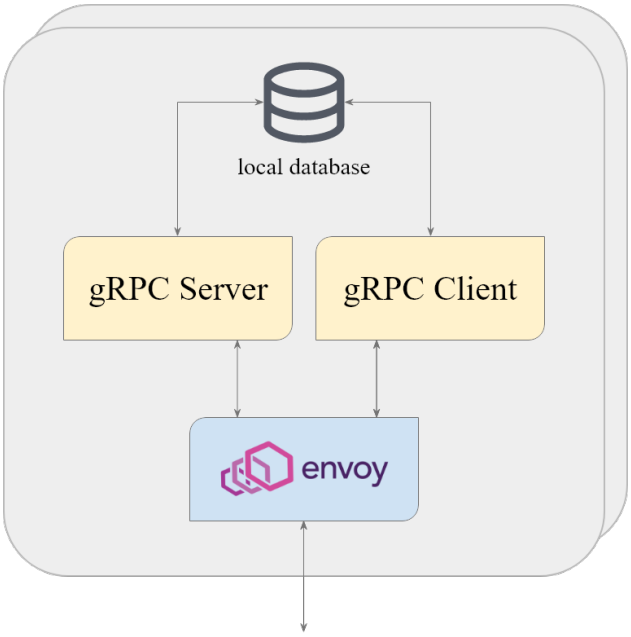

Figure 3.1: Schema of a device in a control system built using gRPC and Envoy

Also, Envoy is a participant in Google's Vulnerability Reward Program (VRP), meaning that it is open to all security researchers and will offer rewards for vulnerabilities detected and reported according to Google and Alphabet Vulnerability Reward Program (VRP) Rules [64]. Besides that, Envoy has encouraged volunteers and users to report any security issues or Envoy crash reports using email [65]. For instance, on December 10, 2019, three vulnerabilities in the Envoy proxy were made public; one of which was categorized as "high severity" and two as "medium severity". On the same day, Envoy released a new version fixing the issues [66]. Hence, not only the internal communications between the gRPC client and gRPC server with Envoy within a device is secure, but also the security of external communications between devices is being managed by Envoy.

3.4.2 Ice

An open-source object-oriented RPC-based middleware framework called Ice (Internet Communication Engine) has emerged as a prominent solution to the problem of communicating in distributed systems, as outlined in the literature review section. In order to provide an explanation of the control system schema that makes use of this framework, we must first define all devices in terms of Ice terminology.

According to Ice terminology [67], a device can be defined as a combination of an *Ice object*, a set of *servants*, one or more *object adapters*, and a number of *proxies*. Ice objects are abstract concepts that can respond to other devices' requests. They are equipped with at least one *interface* and globally unique *object identity*. Interfaces represent a set of named *operations* supported by an object. In addition to a *return value*, each operation has zero or more *parameters*. Each parameter and return value has its own type. Known as a servant, an artifact on each device provides behavior for operation invocations from other devices. A proxy is a representation of an Ice object from another device that allows a device to make requests to the other devices. Clients communicate with a server through an object adapter, a server-side component that provides proxies. Moreover, object adapter routes incoming requests to appropriate methods of the servant representing the target Ice object of the request. In Figure 3.2, we have illustrated the structure of an Ice-based device in a control system that is divided into two components: client-side and server-side.

Ice supports plug-ins that let you add new features to your distributed system without changing its source code. To ensure the integrity of data and verify the identity of the parties performing the communication, the IceSSL plug-in can be installed on devices as a security

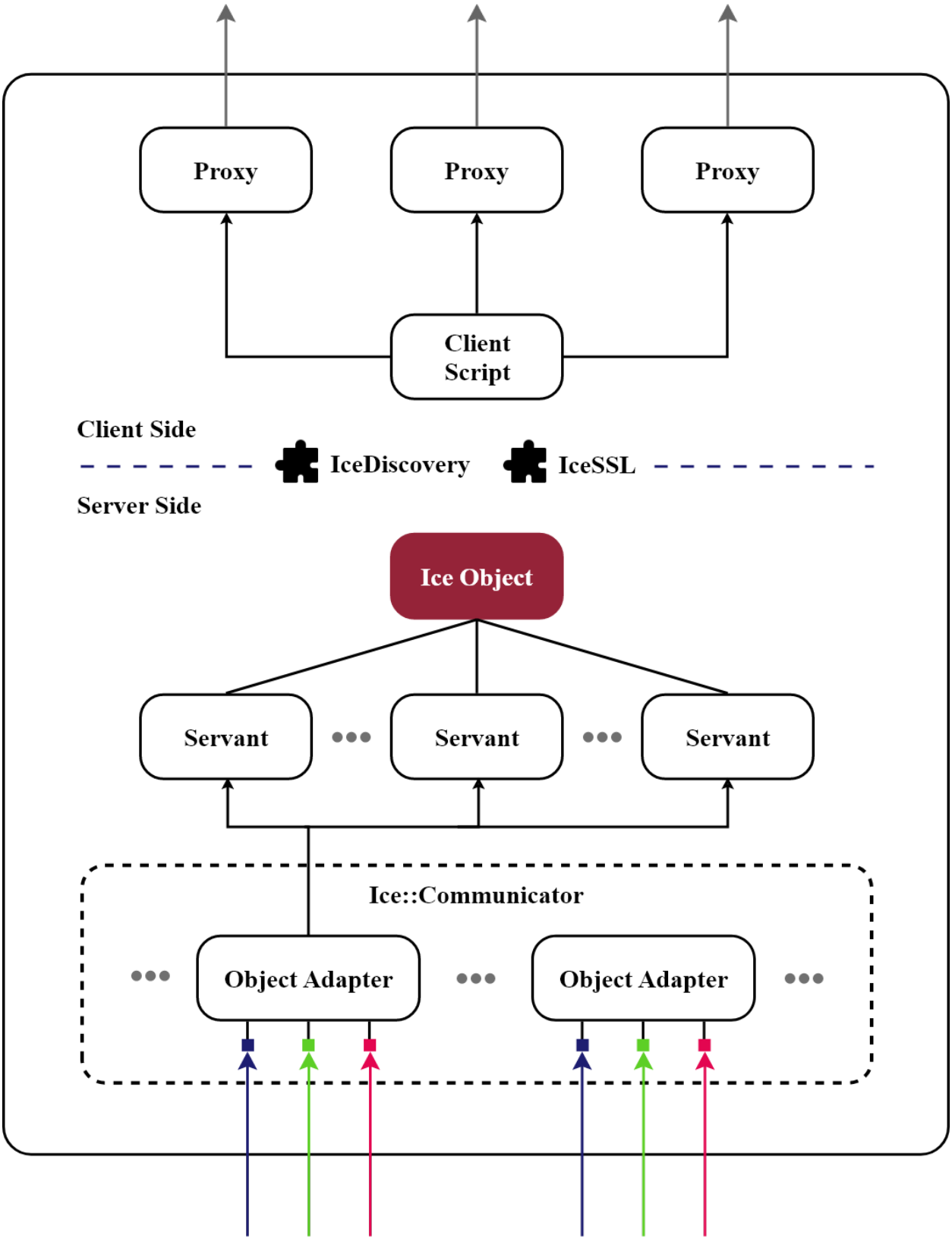

Figure 3.2: Schema of a device in an Ice-based control system

measure. These features are provided by IceSSL through the use of the Transport Layer Security (TLS) protocol [68].

For devices to be able to dynamically locate and communicate with each other, an Ice plug-in called IceDiscovery can be used. IceGrid is an alternative plug-in that provides more features, but since it is associated with a centralized database like Tango, we do not use it to avoid repeating the same error in our solution and keep the solution as light-weight as possible. IceDiscovery utilizes a type of proxy known as indirect proxy in order to achieve its goals. The object's identity is included in all indirect proxies, but some also include an adapter identifier. A location service implements an Ice object called a *locator*. A locator is responsible for transcribing the information provided by an indirect proxy into an endpoint. A custom locator implementation is installed with IceDiscovery to enable discovery via UDP multicast [69].

A control system that has been implemented using Ice allows devices to communicate with each other by sending protocol messages over a medium known as a *transport*. By default, TCP, UDP, and WebSocket transports are supported, while SSL, Bluetooth, and iAP can be enabled by installing their corresponding Ice plug-ins. Hence, we can explain the communication life cycle between devices. Following our understanding of the components of an Ice-based device, we can discuss how they interact with each other.

An indirect proxy should be used when a device (e.g. client) is trying to make an RPC call from another device (e.g. server). IceDiscovery performs a discovery procedure when using an indirect proxy for the first time in order to determine the proxy server that must be provided to the client's location service. Through client's location service, the client caches the endpoints for the object adapter that may be utilized to directly communicate with

the server over the server object adapter's supported transports. As a result of the server's object adapter, a request is received through an endpoint and mapped to the corresponding method of the servant that exists as the manifestation of the requested Ice object. By using the current request's id, the servant will be able to send a Reply protocol message to the client.

For the control system to work with an event management system that can function as an alarm system, such as Tango PANIC [70], the IceStorm service can be employed. This service is used as a publish-and-subscribe mechanism to distribute events for Ice applications. Each group of devices, whose work depends on each other, can utilize the same topic on IceStorm to subscribe to others' events and publish their own. Furthermore, IceStorm allows unidirectional links between topics, allowing messages published on one topic to be published on the receiving topics (i.e. topics linked to a topic with a unidirectional link). This allows the control system to have a hierarchical distributed event handling function, thereby allowing for the management of complex control systems to be flexible. By utilizing redundancy and creating a replica group of IceStorm servers, high availability is ensured in IceStorm.

3.5 Case Study

The Herzberg Astronomy and Astrophysics Research Centre (HAA) of the National Research Council (NRC) is Canada's premier center for astronomy and astrophysics. It maintains the largest and most powerful observatories in Canada and represents Canada at many of the world's leading astronomical events [71]. One of the recent projects that HAA is working on is a control system to control and position dishes, at the Dominion Radio Astro-

physical Observatory [72]. The control system also manages processing the received signals by receivers, and stores the generated data by devices in a database.

An analysis of our proposed candidate solutions against Tango Controls is done by implementing a component of the control system, called Antenna Controller, using both candidates. Figure 3.3 illustrates the software context diagram of the Antenna Controller in the observatory system.

Internal to each Antenna Controller software are two DriveAxis instances (ANT-AST-MTR), corresponding to the two axes of motion for the antenna (latitude and longitude).

The Antenna Controller converts on-sky coordinates into motor encoder coordinates and performs pointing corrections and coordinate transformations. The DriveAxis are then responsible for actual interfacing with the motor control and status hardware. The latitude axis (ANT-AST-MTR(lat)) is the axis that controls pointing in the fundamental plane (elevation or declination) and the longitude axis (ANT-AST-MTR(lon)) controls pointing in

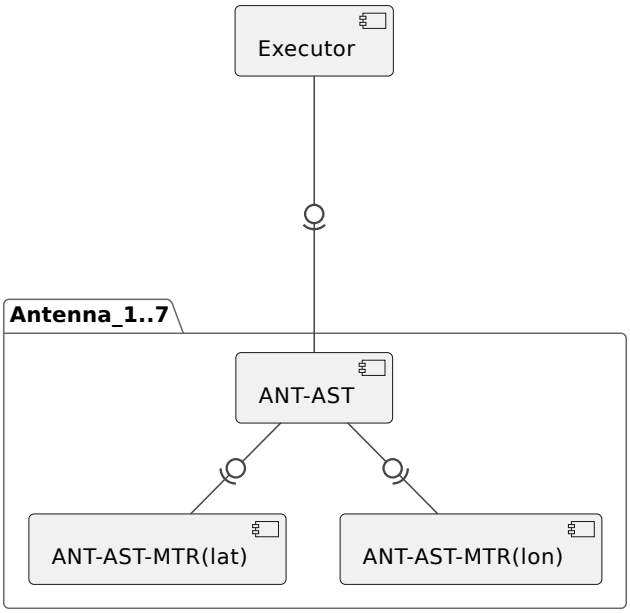

Figure 3.3: Software context diagram of the Antenna Controller

the primary direction (azimuth or hour angle).

Moreover, the executor is the client that maintains a constant connection to the Antenna Controller to carry out commands and monitor the status of the antenna. The ANT-AST is the server that controls the position and velocity of a single antenna by transforming antenna commands into individual commands for the two motors (DriveAxis) that drive the antenna axis. In this section, we explain the requirements for the Antenna Controller component.

3.5.1 Initial Configuration

In order for the Antenna Controller to transition from an initial state to a ready state, the executor will need to send it an initial configuration. The initial configuration is a set of parameters and values which remain constant during the operation of the Antenna Controller (i.e. soft position limits, pointing model coefficients).

3.5.2 Antenna Motion Control

In most situations, querying or sending a command to the Antenna Controller leads to a coordinate transformation followed by a query or command to the DriveAxis interfaces. If the coordinates in the command are not the same coordinate system as the axes of the DriveAxis, they will be transformed to the coordinate system of the axes.

The pointing correction model will then be applied to transform the real (on-sky) coordinates to the encoder (motor device) coordinates.

A sequence diagram is presented in Figure 3.4. This diagram illustrates the flow of requests/responses through the Antenna and DriveAxis interfaces. Thus, implementing location and/or velocity tracking can be achieved in a number of ways as follows:

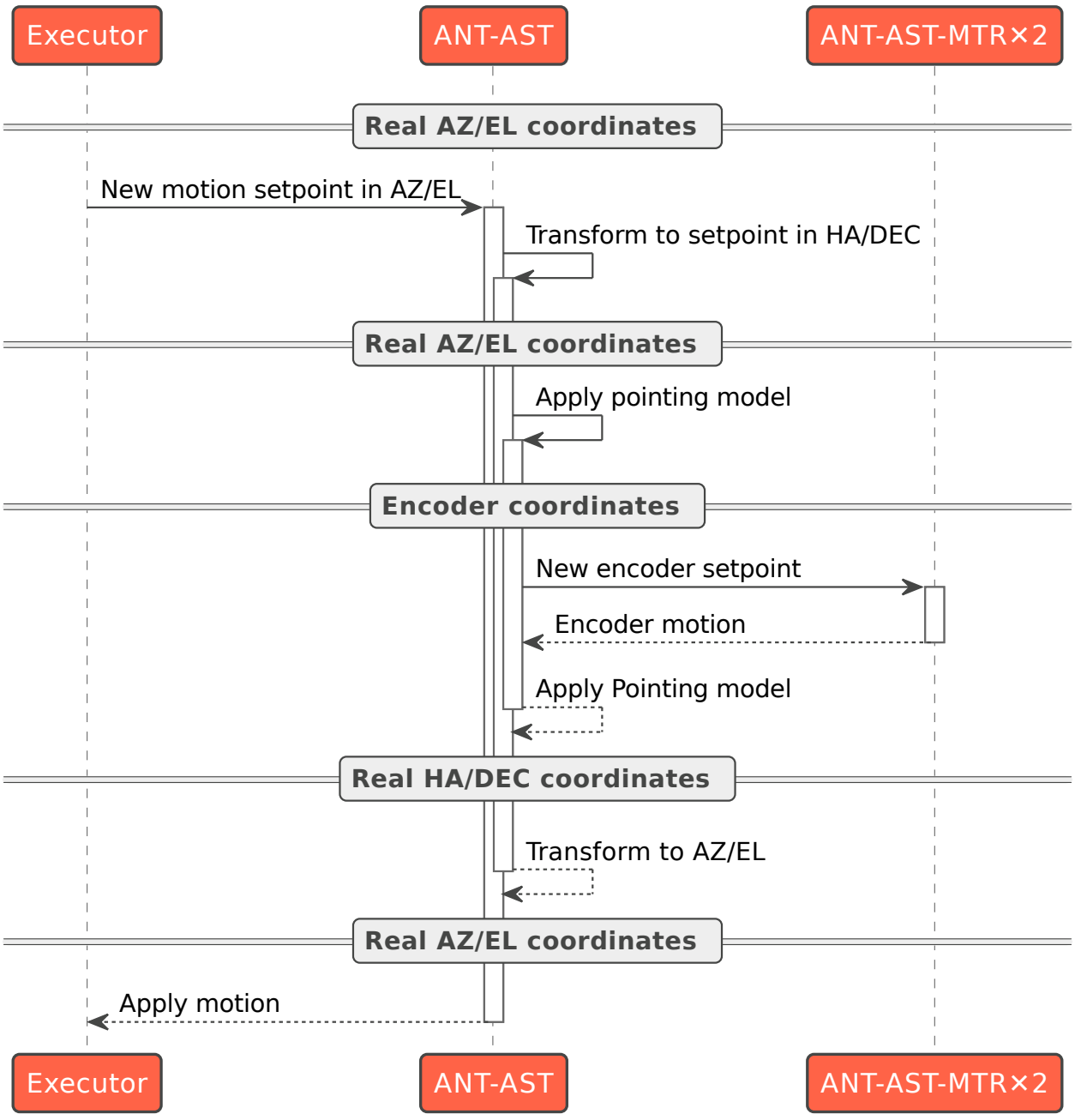

Figure 3.4: Sequence diagram of message flow between the Antenna and DriveAxis interfaces

- Encoder coordinates are written directly to the underlying motor hardware within each ANT-AST-MTR instance if the hardware supports position and/or velocity tracking.

- Encoder coordinates are setpoints of a software PID loop within each ANT-AST-MTR instance. The PID loop can track position or velocity by feeding control actions to the motor hardware.

- Linear Quadratic and other space-state style tracking within the ANT-AST instance for optimizing control of both axes. The outputs of these models can feed the position, velocity, or manual control actions of each ANT-AST-MTR.

3.5.3 DriveAxis Motion Control

The Antenna Controller receives commands in the form of setpoints for the antenna's position or velocity. In order for the DriveAxis to move to the commanded position or velocity, the Antenna Controller will transpose these coordinates into the coordinate system of the two DriveAxis instances. As soon as the DriveAxis has a setpoint, a thread managing the communication to the Antenna Interface Unit (AIU) continually reads the motor encoders and updates the motor drive parameters in order to track to the setpoint.

The Executor can command the Antenna Controller to track a target by continuously updating the setpoint with the up-to-date position of the target. When new setpoints are received, they are compared to the current encoder value, and the control thread adjusts the velocity of the motors to close the error gap between the setpoint and the actual position. The activity diagram of the communication between Antenna Controller and Antenna Interface

Unit is presented in Figure 3.5.

3.5.4 Antenna Interface Unit: The implemented component

The AIU is controlled by an optically isolated RS-422 interface and can also be operated by RS-232. The interface is controlled by a computer system sending a serial command byte followed by any data bytes that are required. A multi-drop communication link is possible with the RS-422 serial communication option for the AIU. The AIU provides support for six

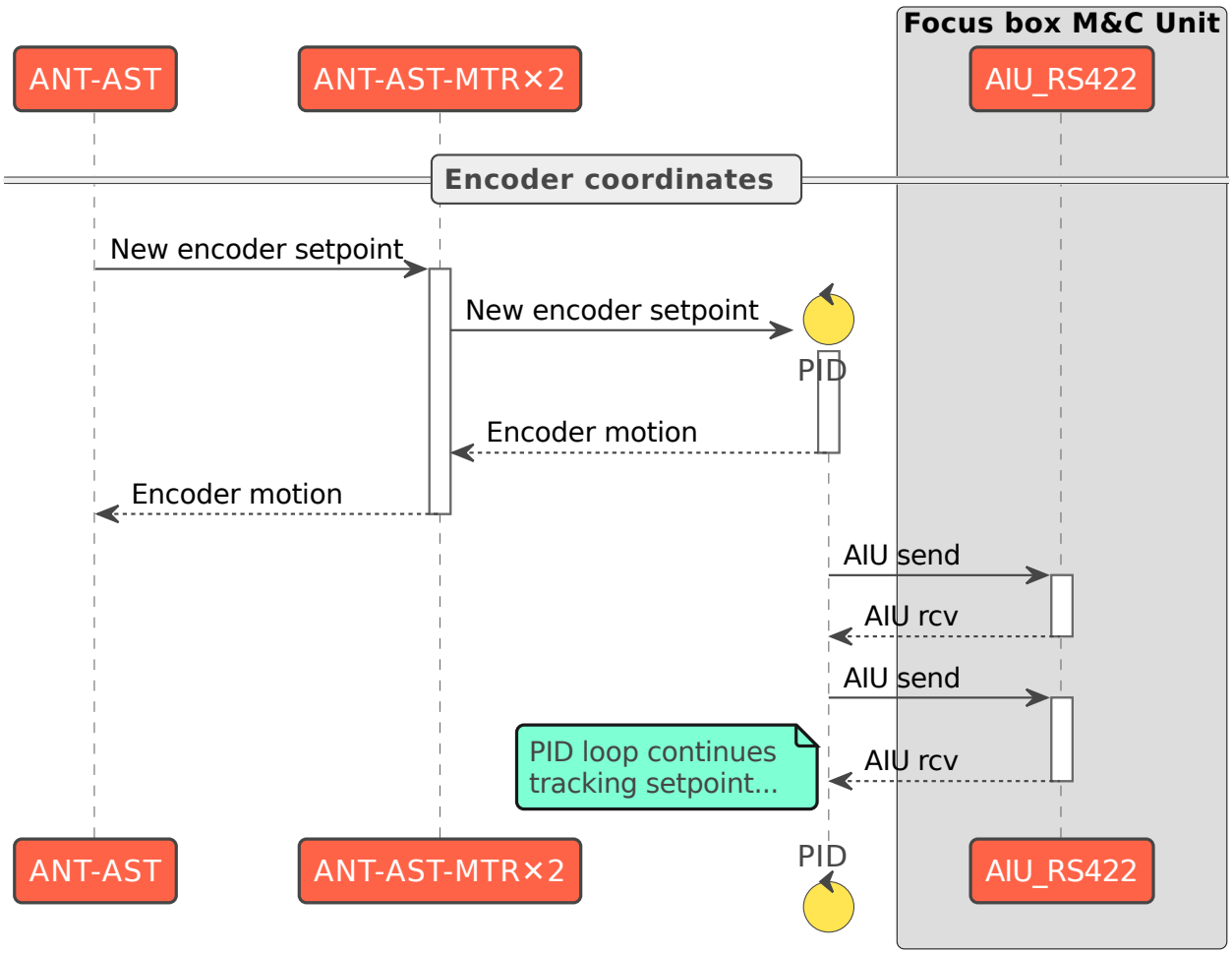

Figure 3.5: Activity diagram of communication between Antenna Controller and Antenna Interface Unit

commands as follows:

- Function 0: Reads the antenna data; syncros, limits, status, and mode.
- Function 1: Writes the motor command; turns motors on for a given time.
- Function 2: Reads from a memory location and returns back the value.
- Function 3: Writes a given byte to a memory location.
- Function 4: Echo test command; Returns a value equal to the antenna address.
- Function 5: Power command; Toggles D7 of the motor direction byte.

We implemented the AIU using three different libraries, including Tango Controls, gRPC and Envoy (Proposed Candidate number 1), and Ice (Proposed Candidate number 2). The AIU has not been connected to an antenna during pre-deployment; therefore, a simulator has been developed in order to simulate the behaviour of the antenna in response to commands. C++ 11 was used as the programming language in order to implement AIU using the three libraries to avoid the ramifications that would arise from using different programming languages. In the following section, we will evaluate the three mentioned implementations.

3.6 Evaluation

The source code of the three implementations named in section 4 e.g. Tango Controls, gRPC, and Ice allows us to evaluate the candidate approaches according to Tango Controls, by utilizing source code software quality analysis techniques. In a recent study, using empirical research, Nilsson [73] has examined the existing tools for internal quality assessment of

software. Among 130 quality assessment tools available, six have been selected by the author based on criteria such as their ability to integrate with IDEs, version control systems, continuous integration and issue tracker systems. The six selected tools were chosen also based on their support for 18 scientifically validated internal quality metrics that were validated by at least two studies in Nilsson's study. In evaluating the six chosen tools, QA-C, Understand, CPPDepend, SonarQube, Eclipse Metrics, and Source Monitor, the results indicate that Understand, QA-C, and SonarQube scored the highest scores respectively. Of the available tools, SonarQube is the only one (at the time of writing this paper) that provides scores for Reliability, Maintainability, Security, and Complexity of the source code. Hence, we chose to use the premium version of this tool, which is equipped with all the features necessary to evaluate our implementations.

In all three implementations, i.e. using Tango Controls, Ice, and gRPC, a tool automatically generated the header files required for the device to be developed. Pogo is a graphical user interface (GUI) used in Tango Controls for the definition of device commands, attributes, and properties. Pogo is able to generate skeleton files, i.e. header files. With gRPC, *protoc* was used to create header files from *.proto* files, which contain messages and service definitions. Like gRPC, Ice uses a compiler called *slice2cpp* to generate header files from *.slice* (Specification Language for Ice) files that encompass modules, interfaces and services. In our evaluation we considered both the auto-generated files and the source codes developed manually for each implementation. In the next section, we will discuss the results gathered from running SonarQube analysis on the implementations.

3.7 Quantitative Results

SonarQube produced various measures with respect to reliability, security, maintainability, size, and complexity for all three of our implementations. Based on the reliability and security assessments, all three implementations received an A rating. In the following subsections, we will discuss the remaining measures.

3.7.1 Maintainability

SonarQube metric definitions [74] define Technical Debt as the effort necessary to fix all Code Smells. Values in days are assumed to be based on an 8-hour work day. Technical debt ratio refers to the ratio of the cost of developing software to the cost of fixing it. These measures are presented in Table 3.1 for each of the three implementations. The maintainability metrics for Ice were lowest,

whereas the Technical Debt and the Technical Debt Ratio were highest for gRPC and Tango Controls.

Table 3.1: Maintainability Measures of The Implementations

Implementation	**Technical Debt**	**Technical Debt Ratio**
gRPC	11 Days	3.2%
Ice	7 Hours 31 Minutes	0.7%
Tango Controls	3 Days 5 Hours	4.5%

3.7.2 Complexity

Cyclomatic Complexity is measured by counting the number of paths through the code. The complexity counter is incremented whenever the control flow of a function is split. Cognitive Complexity refers to the difficulty of understanding the control flow of a program. Campbell has provided a detailed explanation of this metric [75]. Analysis of the results state that Ice achieved the least degree of cyclomatic and cognitive complexity, however gRPC exhibited the greatest degree of cyclomatic complexity, and Tango Controls demonstrated the least degree of understandability. A numerical analysis of the complexity metrics for the implementations is presented in Table 3.2.

3.7.3 Size

Four metrics have been collected from the analysis of the results regarding size. Lines of code can be described as the number of physical lines that contain at least one character. This character does not belong to whitespace, a tabulation, or a comment. Classes encompass the entire set of classes, including interfaces, enums, annotations, and nested classes. The

Table 3.2: Complexity Measures of The Implementations

Implementation	**Cyclomatic Complexity**	**Cognitive Complexity**
gRPC	605	159
Ice	105	116
Tango Controls	255	173

other two metrics count the number of functions and files for each implementation. The implementation using gRPC resulted in the largest size across three of the four size metrics, i.e. lines of code, classes, and functions. Conversely, the implementation by Ice achieved the lowest number of files, functions and classes. Table 3.3 outlines the results of the size metrics calculated for the three implementations.

3.7.4 Results Conclusion

The scoring for Ice was favorable in seven of the eight measures, making it the best candidate to replace Tango Controls based on maintainability and complexity. In contrast, the other candidate, gRPC, achieved the poorest results in half of the metrics, and therefore, cannot be considered a suitable alternative to Tango Controls.

3.8 Empirical Analysis

The purpose of this section is to summarize the empirical analysis conducted using the three candidates to implement the case study. Tango Controls offers a graphical user interface

Table 3.3: Size Measures of The Implementations

Implementation	**Lines of Code**	**Classes**	**Functions**	**Files**
gRPC	5531	59	516	5
Ice	2239	7	56	3
Tango Controls	1315	22	158	7

known as Pogo to generate Skeleton codes for the device server, as discussed in the Evaluation section. When using this tool, novice users are faced with the challenge of selecting between device properties, read-only device attributes, write-only device attributes, and read/write device attributes. The limited number of input arguments for Tango commands presents a further challenge to overcome. Specifically, Tango Controls allow only one input and output argument per command [76]. Though the argument may be an array of a specific data type, since a command may output or input data of different types, the developer must deal with data conversions manually in order to fit the data into the array.

This means that during the development process more code will be added, which could affect the maintainability of the implementation. Moreover, the implementation of the device server takes place within one of the automatically generated source code files by Pogo within the defined protected regions of the code. As a result of the presence of both automatically generated and manually added codes in a file, maintainability issues may arise since developers will have to locate code blocks that have been added manually. With gRPC and Ice, the auto-generated files are kept apart from the implementation source code that is developed manually.

On the other hand, the implementation procedure using gRPC and Ice was similar, since both are RPC-based frameworks. Both frameworks require the supported services to be defined with a specific syntax in a file that is used to generate the skeleton (header) files. In the case of gRPC, however, the use of a supported proto data type for the input or output arguments of the RPC is not permitted. In other words, basic data types should be defined as messages before being used in RPC arguments. The result is an extra amount of work and the creation of multiple messages with a single message field. In addition, each field in

the message definition is uniquely numbered in the Proto language [77]. The field number is used to identify each field in the binary format of a message, and should not be changed once your message type is in place. This number should be assigned manually, so the developer's involvement is required. In contrast, Ice did not experience the difficulties noted above in defining RPCs in the Slice language.

3.9 Threats To Validity

In our methodology focused only on open-source candidates, but there may also be commercial solutions that may be considered as Tango Controls alternatives. In addition, while we used static source code analysis to compare the candidates based on maintainability, complexity, and size, more advanced metrics related to performance, such as load testing, could be used to further examine the discussed candidates. Furthermore, considering the size and complexity of the device to which the candidate solutions were applied, the concluded results may or may not be applicable to larger and more complex devices that form a control system.

3.10 Conclusion & Future Work

This chapter explored two potential middleware frameworks to replace Tango Controls by reviewing relevant literature and gathering trends in solutions that are currently being used in similar settings. With gRPC and Ice as the candidates, we conducted a real-world case study at the Dominion Radio Astrophysical Observatory using these candidates and Tango Controls. We used SonarQube to conduct a static analysis of the source code of each

implementation to determine metrics relevant to maintainability, complexity, and size of the source code of each candidate solution. In addition, we discussed the empirical analysis of those implementations from different perspectives. Considering the results of the SonarQube analysis and our own empirical analysis, Ice is the most appropriate replacement for Tango Controls. In the future, we may include commercial off-the-shelf middleware frameworks in our comparison, as well as perform performance and scalability tests, as well as measure overhead for each candidate solution.

Chapter 4

Conclusion and Future Work

4.1 Summary and Conclusion

This thesis elaborated on a process to analyze Tango Controls' software quality threats. We proposed alternative solutions to replace Tango Controls and introduced the superior.

The overall process started with creating an evaluation checklist based on the SCADA intrinsic requirements. The checklist was then used to analyze Tango Controls' mechanisms by scrutinizing its documentation. The mechanisms and features were examined by considering two general types of control systems, distributed and centralized. The analysis led to the identification of three drawbacks that violated our checklist items. Chapter 2 discussed the process up to this point.

In Chapter 3, we continued the research progress by proposing two candidate solutions combined based on the CBSD approach. Our candidates were gRPC along with Envoy, and Ice. We discussed the potential of each candidate solution to cover Tango Controls' disadvantages. Since we showed both candidates could cover the drawbacks, we needed to evaluate them against Tango Controls further. Hence, we implemented a real-world control system module at DRAO using candidate solutions and Tango Controls.

We relied on SonarQube's static source code analysis results to evaluate the implementation in terms of code complexity, maintainability of the code, and their sizes. According to the quantitative results, Ice outperformed Tango Controls and the combination of gRPC and

Envoy in 7 out of 8 measures. It was only on the lines of code that Ice could not compete with other alternatives.

Furthermore, we empirically analyzed the implementation from a developer's perspective to highlight each candidate's strengths and weaknesses. The following are the highlights of each candidate:

- Tango Controls
  - Issues
    * Confusing GUI options for novice users without a guideline.
    * Limited input and output arguments.
    * User-written code is mixed with auto-generated code.
  - Impact
    * Lowering code maintainability
    * Increased code complexity
- gRPC+Envoy
  - Issues
    * Lack of support for proto data types in RPC arguments
    * Requiring manual input from the developer for field numbers.

- Impact
    - Increased lines of code

According to our quantitative results and empirical analysis, Ice is superior to gRPC+Envoy and Tango Controls.

4.2 Limitations

The initial constraint of our study pertains to the sources we utilized to compile our evaluation checklist. While scholarly references offer a scientific foundation for conceptual understanding, incorporating the viewpoints of practitioners would have strengthened our checklist's reliability. By engaging with control system practitioners from institutions like DRAO and global observatories like ALBA in Spain, a more triangulated perspective could have been achieved. This avenue would have provided insights into real-world scenarios and diversified our control systems scenarios.

The subsequent limitation we encounter revolves around the resources employed to identify software quality threats within Tango Controls. Our reliance on its documentation stems from its status as a primary guideline shaped by the OSS developer community. This approach aimed to minimize potential misinformation by tapping directly into the expertise of OSS developers. However, broadening the spectrum of information sources during the evaluation phase could have exposed additional issues. Such diversification would have mitigated any inherent bias towards Tango Controls' content.

The formulation of our alternative solutions rested on references to OSS libraries, a choice rooted in the widespread adoption of OSS across industrial and governmental domains. How-

ever, exploring Commercial Off-The-Shelf (COTS) alternatives might have yielded different candidates. In the context of static source code analysis, we drew from a prior study that evaluated various tools, elevating SonarQube – our chosen tool – to their elite selections. The objective selection of SonarQube was guided by existing research and the comprehensiveness of its metrics. While our analysis encompassed three distinct measurement dimensions for each implementation, further comparisons could delve into advanced metrics like load testing.

However, it's important to note that our findings might not readily extend to expansive control systems with multifarious modules, given our focus on a rudimentary control system module during implementation. Additionally, the evaluation's foundation in the researcher's developed code introduces the potential for researcher-based expertise influencing quantitative outcomes. Nevertheless, this influence remains minimal due to the uniform familiarity with all alternatives and Tango Controls held by the researcher, thus mitigating the risk of bias.

4.3 Future Work

This study highlighted a notable concern: the considerable time and resource investment required for executing an assessment of OSS quality. This concern is particularly relevant for small to medium-sized businesses.

In recent years, the adoption of Deep Learning techniques to address diverse text classification assignments has gained substantial traction. Tasks such as question answering, sentiment analysis, topic analysis, natural language inference, and news categorization have all witnessed a surge in the application of Deep Learning methodologies. In particular, Nat-

ural Language Inference has garnered attention for its capacity to enable machines to deduce conclusions based on an understanding of real-world principles, extending beyond explicit textual content [78].

Numerous online resources are at our disposal to gather information about OSS projects. GitHub, a prominent platform hosting a plethora of open-source projects, serves as a repository of valuable insights. It offers extensive data on pull requests, forked repositories, stars, tags, and incorporates social coding functionalities [79]. Leveraging this information, valuable software-related metrics can be inferred, aiding in assessing maintenance activities within OSS projects [80].

In the realm of developer engagement, Stack Overflow stands as a prime illustration of the escalating popularity of community-driven question-and-answer (Q&A) platforms. Boasting over a hundred million monthly visitors and a user base exceeding 21 million, Stack Overflow underscores the growth in the appeal of such platforms among developers [81]. Given the vast wealth of information amassed on this platform, researchers have efficiently mined its resources for an array of Software Engineering tasks [82–85].

Consequently, the trajectory of this research points towards the expansion and generalization of the findings presented in this thesis. This will be achieved by introducing a natural language processing (NLP)-powered tool that systematically processes various online information sources related to an OSS library. The ultimate goal of this tool is to furnish a comprehensive software quality report. Moreover, this tool's potential is not limited to reporting; it has the capacity to evolve into a dynamic recommendation system for open-source software. This recommendation system would not only offer insightful suggestions but also possess the capability to adapt and refine its recommendations over time, driven by ongoing

monitoring of pertinent information sources.

In conclusion, the fusion of deep learning techniques, comprehensive online resources, and the development of an intelligent NLP-powered tool marks a promising direction for the future. This trajectory has the potential to reshape how businesses approach OSS assessment and adoption, fostering more informed decision-making, optimizing resource allocation, and ultimately contributing to the advancement of software quality and innovation. As this research evolves into actionable tools, the impact on both small and medium-sized enterprises and the wider software development community could be transformative.

Bibliography

[1] "Tango REST API — Tango Controls 9.3.4 documentation." [Online]. Available: https://tango-controls.readthedocs.io/en/latest/development/advanced/rest-api.html

[2] N. A. K. Umm-e Laila, A. Arfeen, and S. Hassan, "TRENDS OF OPEN SOURCE SOFTWARE IN MISSION CRITICAL ITS SERVICES INFRASTRUCTURES ADOPTION IN LOCAL ENVIRONMENT."

[3] V. Lenarduzzi, D. Taibi, D. Tosi, L. Lavazza, and S. Morasca, "Open Source Software Evaluation, Selection, and Adoption: A Systematic Literature Review," *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, pp. 437–444, 8 2020.

[4] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, "The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation," *IFIP International Federation for Information Processing*, vol. 275, pp. 237–248, 2008. [Online]. Available: https://link.springer.com/chapter/10.1007/978-0-387-09684-1_19

[5] A. Imani, M. Moshirpour, and L. Belostotski, "Checklist-based Software Quality Evaluation of Tango Controls," in *Proceedings - 2021 International Conference on Information Systems and Advanced Technologies, ICISAT 2021*, 2021.

[6] R. M. Van Der Knijff, "Control systems/SCADA forensics, what's the difference?" *Digital Investigation*, vol. 11, no. 3, pp. 160–174, 9 2014.

[7] A. Daneels and W. Salter, "What is SCADA?" 1999. [Online]. Available:

https://accelconf.web.cern.ch/ica99/papers/mc1i01.pdf

[8] I. Morsi and L. M. El-Din, "SCADA system for oil refinery control," *Measurement*, vol. 47, no. 1, pp. 5–13, 1 2014.

[9] O. Barana, P. Barbato, M. Breda, R. Capobianco, A. Luchetta, F. Molon, M. Moressa, P. Simionato, C. Taliercio, and E. Zampiva, "Comparison between commercial and open-source SCADA packages—A case study," *Fusion Engineering and Design*, vol. 85, no. 3-4, pp. 491–495, 7 2010.

[10] "About us - TANGO Controls." [Online]. Available: https://www.tango-controls.org/about-us/#History

[11] "Why choose Tango Controls ? - TANGO Controls." [Online]. Available: https://www.tango-controls.org/why-tango-controls/

[12] S. Rubio-Manrique, G. Cuní, D. Fernández-Carreiras, C. Pascual-Izarra, D. Roldán, and E. Al-Dmour, "Unifying all TANGO control services in a customizable graphical user interface," *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*, 1 2016. [Online]. Available: http://inis.iaea.org/Search/search.aspx?orig_q=RN:51100788

[13] "WELCOME TO ALBA — en." [Online]. Available: https://www.albasynchrotron.es/en/about/welcome

[14] "Welcome to Tango Controls documentation! — Tango Controls 9.3.4 documentation." [Online]. Available: https://tango-controls.readthedocs.io/en/latest/index.html

[15] P. P. Goryl and M. Liszcz, "TOWARDS IMPROVED ACCESSIBILITY OF THE TANGO CONTROLS *," 2019. [Online]. Available: https://www.docslikecode.com

[16] "Classes Catalogue - TANGO Controls." [Online]. Available: https://www.tango-controls.org/developers/dsc/

[17] R. Bourtembourg, S. James, J. L. Pons, P. Verdier, G. Cuni, S. Rubio-Manrique, G. A. Fatkin, A. I. Senchenko, V. Sitnov, L. Pivetta, and others, "Pushing the Limits of Tango Archiving System using PostgreSQL and Time Series Databases," in *17th Biennial International Conference on Accelerator and Large Experimental Physics Control Systems*, 2019.

[18] M. Drochner, L. Fleischhauer-Fuss, H. Kleines, M. Wagener, S. v. Waasen, and F. Z. Jülich, "Neutron scattering instrument control system modernization-front-end hardware and software adaption problems," 2015.

[19] D. Bolkhovityanov and P. Cheblakov, "A Comparative Analysis of the Architecture of Control Systems of Physical Research Facilities," *Physics of Particles and Nuclei Letters 2020 17:4*, vol. 17, no. 4, pp. 571–573, 7 2020. [Online]. Available: https://link.springer.com/article/10.1134/S1547477120040123

[20] R. Barcelos and G. Travassos, "Evaluation Approaches for Software Architectural Documents: a Systematic Review." 9 2006, pp. 433–446.

[21] Y. Chen, X. Yan, and A. A. Khan, "A Novel Reliability Assessment Method Based on the Effects of Components," in *Proceedings - 19th IEEE International Conference*

on Software Quality, Reliability and Security, QRS 2019. Institute of Electrical and Electronics Engineers Inc., 7 2019, pp. 69–76.

[22] C. Alcaraz and J. Lopez, "Analysis of requirements for critical control systems," *International Journal of Critical Infrastructure Protection*, vol. 5, no. 3-4, pp. 137–145, 12 2012.

[23] K. Stouffer, J. Falco, K. Scarfone, and others, "Guide to industrial control systems (ICS) security," *NIST special publication*, vol. 800, no. 82, p. 16, 2011.

[24] T. Frank, M. Merz, K. Eckert, T. Hadlich, B. Vogel-Heuser, A. Fay, and C. Diedrich, "Dealing with non-functional requirements in distributed control systems engineering," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2011.

[25] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 1 2004.

[26] B. Penzenstadler, "Towards a Definition of Sustainability in and for Software Engineering," *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13*, 2013.

[27] "Overview of Tango Controls — Tango Controls 9.3.4 documentation." [Online]. Available: https://tango-controls.readthedocs.io/en/latest/overview/overview.html

[28] "Glossary — Tango Controls 9.3.4 documentation." [Online]. Available: https://tango-controls.readthedocs.io/en/latest/reference/glossary.html

[29] J. McCall, P. Richards, and G. Walters, "Factors in software quality. volume i. concepts and definitions of software quality," vol. I, 1977. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA049014

[30] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 592–605.

[31] R. Grady, *Practical software metrics for project management and process improvement*, 1992. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/140207

[32] J. P. Miguel, D. Mauricio, and G. Rodriguez, "A Review of Software Quality Models for the Evaluation of Software Products," *International Journal of Software Engineering & Applications*, vol. 5, no. 6, pp. 31–53, 12 2014. [Online]. Available: https://arxiv.org/abs/1412.2977v1

[33] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, 2002, pp. 547–550.

[34] V. R. Westmark, "A definition for information system survivability," *Proceedings of the Hawaii International Conference on System Sciences*, vol. 37, pp. 4827–4836, 2004.

[35] R. D'Andrea and G. E. Dullerud, "Distributed Control Design for Spatially Interconnected Systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 9, pp. 1478–1495, 9 2003.

[36] "The Tango controlled access system — Tango Controls 9.3.4 documentation."

[Online]. Available: https://tango-controls.readthedocs.io/en/latest/administration/services/access-control.html

[37] "HAProxy version 2.3.10 - Starter Guide." [Online]. Available: http://cbonte.github.io/haproxy-dconv/2.3/intro.html

[38] N. Dattatreya Nadig, "Testing Resilience of Envoy Service Proxy with Microservices," *DEGREE PROJECT IN TECHNOLOGY*, 2019.

[39] H. Alrubaye, D. Alshoaibi, E. Alomar, M. W. Mkaouer, and A. Ouni, "How Does Library Migration Impact Software Quality and Comprehension? An Empirical Study," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Cham, 12 2020, vol. 12541 LNCS, pp. 245–260. [Online]. Available: http://link.springer.com/10.1007/978-3-030-64694-3_15

[40] C. Szyperski, *Component software: beyond object-oriented programming*. New York : Harlow, England ; Reading, Mass.: ACM Press ; Addison-Wesley, 1997. [Online]. Available: //catalog.hathitrust.org/Record/003963284http://hdl.handle.net/2027/mdp.39015040372081

[41] P. Chatzipetrou, E. Papatheocharous, K. Wnuk, M. Borg, E. Alégroth, and T. Gorschek, "Component attributes and their importance in decisions and component selection," *Software Quality Journal*, vol. 28, no. 2, pp. 567–593, 6 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11219-019-09465-2

[42] M. Morandini, A. Siena, and A. Susi, "Risk Awareness in Open Source Component

Selection," *Lecture Notes in Business Information Processing*, vol. 176 LNBIP, pp. 241–252, 2014. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-06695-0_21

[43] S. Butler, J. Gamalielsson, B. Lundell, C. Brax, A. Mattsson, T. Gustavsson, J. Feist, B. Kvarnström, and E. Lönroth, "Considerations and challenges for the adoption of open source components in software-intensive businesses," *Journal of Systems and Software*, vol. 186, p. 111152, 4 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221002442?via%3Dihub

[44] D. Spinellis, "How to Select Open Source Components," *Computer*, vol. 52, no. 12, pp. 103–106, 12 2019.

[45] F. Khan, M. Tahir, F. Arif, M. Babar, and S. Khan, "Framework for Better Reusability in Component Based Software Engineering," *Journal of Applied Environmental and Biological Sciences*, vol. 6, pp. 77–81, 1 2016.

[46] O. Al-Debagy and P. Martinek, "A Comparative Review of Microservices and Monolithic Architectures," *18th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2018 - Proceedings*, pp. 149–154, 11 2018.

[47] R. Ravanmehr and A. Jafarzadeh, "INO340 telescope control system: Software architecture and development," vol. 9152, 2 2014, p. 91521Q.

[48] K. Kirill, "Software Architecture of Control System for Heterogeneous Group of Mobile Robots," *Procedia Engineering*, vol. 100, no. January, pp. 278–282, 1 2015.

[49] J. Li, N. Wang, Z. Liu, Y. Song, N. Li, L. Xu, and J. Wang, "Trends in Architecture and Middleware of Radio Telescope Control System," *Advances in Astronomy*, vol. 2021, p. 2655250, 2021. [Online]. Available: https://doi.org/10.1155/2021/2655250

[50] M. García-Valls, D. Garrido, and M. Díaz, "Impact of Middleware Design on the Communication Performance," in *International Conference on Green, Pervasive, and Cloud Computing*, vol. 10232 LNCS. Springer, Cham, 2017, pp. 505–519. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-57186-7_37

[51] Y. Li, J. Zhou, L. Guo, Y. Wang, and R. Yi, "Research on Distributed Network Communication Based on ICE Middleware," in *Proceedings of the 2016 6th International Conference on Machinery, Materials, Environment, Biotechnology and Computer*. Atlantis Press, 6 2016, pp. 577–581. [Online]. Available: https://doi.org/10.2991/mmebc-16.2016.124

[52] "[Analyst Report] 2021 Open Source Security and Analysis Report — Synopsys." [Online]. Available: https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

[53] "About gRPC — gRPC." [Online]. Available: https://grpc.io/about/

[54] "Introduction to gRPC — gRPC." [Online]. Available: https://grpc.io/docs/what-is-grpc/introduction/

[55] B. Burns and D. O. Google, "Design patterns for container-based distributed systems."

[56] "Sidecar proxy — Mastering Service Mesh." [Online].

Available: https://learning.oreilly.com/library/view/mastering-service-mesh/9781789615791/fa3c5838-691d-40e9-b582-a34d3a904e4b.xhtml

[57] A. Khatri and V. Khatri, *Mastering Service Mesh: Enhance, secure, and observe cloud-native applications with Istio, Linkerd, and Consul.* Packt Publishing Ltd, 2020.

[58] "The world's lightest, fastest service mesh. — Linkerd." [Online]. Available: https://linkerd.io/

[59] "Envoy Proxy - Home." [Online]. Available: https://www.envoyproxy.io/

[60] "What is Envoy — envoy 1.19.0-dev-4533ea documentation." [Online]. Available: https://www.envoyproxy.io/docs/envoy/latest/intro/what_is_envoy

[61] "Life of a Request — envoy 1.19.0-dev-4533ea documentation." [Online]. Available: https://www.envoyproxy.io/docs/envoy/latest/intro/life_of_a_request

[62] "Authentication — gRPC." [Online]. Available: https://grpc.io/docs/guides/auth/

[63] "Security — envoy 1.19.0-dev-5c8d4d documentation." [Online]. Available: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/security/security

[64] "Google Vulnerability Reward Program (VRP) — envoy 1.19.0-dev-5c8d4d documentation." [Online]. Available: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/security/google_vrp#google-vulnerability-reward-program-vrp

[65] "Security Policy · envoyproxy/envoy." [Online]. Available: https://github.com/envoyproxy/envoy/security/policy

[66] "Recent Vulnerabilities in Envoy Explained, Including Impact to Istio." [Online]. Available: https://www.paloaltonetworks.com/blog/2019/12/cloud-envoy-vulnerabilities/

[67] "Terminology - Ice." [Online]. Available: https://doc.zeroc.com/ice/3.7/ice-overview/ice-architecture/terminology

[68] "IceSSL - Ice." [Online]. Available: https://doc.zeroc.com/ice/3.7/ice-plugins/icessl

[69] "IceDiscovery - Ice." [Online]. Available: https://doc.zeroc.com/ice/3.7/ice-plugins/icediscovery

[70] "PANIC Description — panic documentation." [Online]. Available: https://tango-controls.readthedocs.io/projects/panic/en/latest/description.html

[71] "Herzberg Astronomy and Astrophysics Research Centre." [Online]. Available: https://nrc.canada.ca/en/research-development/research-collaboration/research-centres/herzberg-astronomy-astrophysics-research-centre

[72] "Dominion Radio Astrophysical Observatory research facility." [Online]. Available: https://nrc.canada.ca/en/research-development/nrc-facilities/dominion-radio-astrophysical-observatory-research-facility

[73] M. Nilsson, "A Comparative Case Study on Tools for Internal Software Quality Measures," 2019. [Online]. Available: http://hdl.handle.net/2077/62453

[74] "Metric Definitions — SonarCloud Docs." [Online]. Available: https://docs.sonarcloud.io/digging-deeper/metric-definitions/

[75] G. Ann Campbell, "Cognitive Complexity — An Overview and Evaluation," *Proceedings of the 2018 International Conference on Technical Debt*, 2018. [Online]. Available: https://doi.org/10.1145/3194164.3194186

[76] "Guidelines — Tango Controls 9.3.4 documentation." [Online]. Available: https://tango-controls.readthedocs.io/en/latest/development/device-api/ds-guideline/device-server-guidelines.html

[77] "Language Guide (proto3) — Protocol Buffers — Google Developers." [Online]. Available: https://developers.google.com/protocol-buffers/docs/proto3

[78] S. Storks, Q. Gao, and J. Y. Chai, "Recent Advances in Natural Language Inference: A Survey of Benchmarks, Resources, and Approaches," 4 2019. [Online]. Available: https://arxiv.org/abs/1904.01172v3

[79] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining GitHub for research and education: challenges and opportunities," *International Journal of Web Information Systems*, 2020.

[80] J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects," *Information and Software Technology*, vol. 122, p. 106274, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584920300240

[81] "Empowering the world to develop technology through collective knowledge - Stack Overflow." [Online]. Available: https://stackoverflow.co/

[82] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, "Sentiment Analysis for Software Engineering: How Far Can We Go?" in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 94–104. [Online]. Available: https://doi.org/10.1145/3180155.3180195

[83] S. Wang, N. Phan, Y. Wang, and Y. Zhao, "Extracting API Tips from Developer Question and Answer Websites," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 321–332.

[84] G. Uddin and F. Khomh, "Automatic Mining of Opinions Expressed About APIs in Stack Overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 522–559, 2021.

[85] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment Polarity Detection for Software Development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9546-9

Appendix A

Notification of Acceptance

2023 3rd International Conference on Computer, Control and Robotics (ICCCR 2023)

Shanghai, China | March 24--26, 2023

http://www.icccr.org/

Dear Alireza Imani, Mohammad Moshirpour, Leonid Belostotski and David A Del Rizzo,

First of all, thank you for your concern. The review processes for the paper you submitted to **2023 3rd International Conference on Computer, Control and Robotics (ICCCR 2023)** have been completed. We are delighted to inform you that your manuscript has been accepted for presentation and publication. The reviewers' comments are enclosed.

The conference received submissions from different countries and regions, which were all reviewed by international experts. Based on the recommendations of reviewers and the technical program committees, we are pleased to inform you that your paper identified below has been accepted for publication and oral presentation on conference. You are cordially invited to present the paper orally at ICCCR 2023 which will be held in **Shanghai, China during March 24-26, 2023.** ICCCR 2023 is co-sponsored by IEEE, IEEE Robotics and Automation Society, Shanghai University, and Anhui University of Science and Technology, hosted by School of Mechatronic Engineering and Automation, Shanghai University, and School of Mechanical Engineering, Anhui University of Science and Technology.

Paper ID:	SH2083
Paper Title:	Towards Replacing Tango Controls: A Comparative Empirical Study

Accepted and presented papers will be published in ICCCR 2023 Conference Proceedings, which will be published and submitted to IEEE Xplore. Those papers will be indexed by Ei Compendex, Scopus and other main databases.

Good News: ICCCR 2022 conference proceedings has been archived in IEEE Xplore and indexed by Ei Compendexs and Scopus three month after the conference.

ICCCR 2023 Organizing Committee
Shanghai, China
http://www.icccr.org/

Figure A.1: Chapter 3 Notification of Acceptance