EVALUATION OF DISTRIBUTED-SYSTEM TECHNOLOGIES FOR ALMA COMMON SOFTWARE

SANHUEZA ULSEN, RENATO FULVIO

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE

"EVALUATION OF DISTRIBUTED-SYSTEM TECHNOLOGIES FOR ALMA COMMON SOFTWARE"

RENATO FULVIO SANHUEZA ULSEN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Horst H. von Brand
Profesor Correferente: Mauricio Araya

Julio - 2018

DEDICATION

This thesis is dedicated to my mother, who taught me to strive to the excellence in all aspects of life. It is also dedicated to my father, who always has my back in the direst times.

ACKNOWLEDGMENT

The valuable advice given by my thesis advisors, Horst von Brand and Mauricio Araya, has been a great help in the development and improvement of this document. The hardware and software provided by the UTFSM Computer Systems Research Group for the execution of the experiments was greatly appreciated. I would also like to thank the middleware community, especially the people in the EPICS mailing list and the ZeroC forums who shared with me their knowledge and experience about middleware.

Last but not least, I would like to thank my parents and my godfather. Without their unconditional love and support this work could not be possible.

ABSTRACT

Modern observatories develop distributed software systems to support their operations (e.g., data acquisition, hardware control and system monitoring). The complexity of these systems makes it difficult and even infeasible to build them from scratch so the software developers rely on Off-The-Shelf (OTS) software, such as communication middleware, for the development process. ALMA Common Software (ACS) is an open-source software framework for the development of distributed control systems that is based on CORBA middleware. Unfortunately the CORBA compliant middleware is struggling to keep up with the increasingly demanding requirements of emergent observatories. This dissertation propose an iterative solution, based on the PECA process created by the Software Engineering Institute, for the evaluation and selection of communication middleware alternatives for ACS, so the framework can be used by modern observatories, specifically Cherenkov Telescope Array (CTA). The process described in this work provides several advantages to the organizations working with OTS software. It reduces long-term cost and risk of failure of the system, improves the knowledge about the system and its context, and allows the evaluation team to tailor the process to fit their needs and budget. Furthermore it also contributes to upgrade the reuse maturity of the organization and the tailorability of the process makes it applicable in many contexts such as the evaluation and selection of other key components of the system. Due to resource constraints this work only provides a partial execution of the process, that aims to serve as a guide for further evaluation efforts. Even though no middleware candidate should be selected without a full implementation of the process, this first iteration describes the approach in detail and it generated several results that might be reused in future iterations such as a hierarchical model of the multi-criteria decision-making problem, prioritization of the criteria, filtered pool of middleware candidates, data collection techniques recommendations, latency benchmarking and coupling analysis of the most promising candidates.

***Keywords*— ALMA Common Software; Cherenkov Telescope Array; Communication middleware; Off-The-Shelf software selection; PECA**

GLOSSARY

ACS: ALMA Common Software.

ACTL: Array Control & Data Acquisition

AHP: Analytic Hierarchy Process

ALMA: Atacama Large Millimeter/submillimeter Array

API: Application programming interface

CAP: COTS Acquisition Process

CBD: Component Based Development

CBSS: Component based software system

CCL: Control Command Language

CCM: Container Component Model

CERN: European Organization for Nuclear Research

CMW: Controls Middleware

COA: Component Oriented Architecture

CORBA: Common Object Request Broker Architecture

COTS: Commercial off-the-shelf

CS: Combined-Selection

CTA: Cherenkov Telescope Array

DA: Domain and architecture compatibility

ESO: European Southern Observatory

FR: Functional requirements

FTE: Full-time equivalent

GCS: General COTS Selection

IDL: Interface definition language

INAF: National Institute for Astrophysics

LHC: Large Hadron Collider

LST: Large-sized telescope

MCDM: Multiple-criteria decision-making

MiHOS: Mismatch-Handling aware COTS Selection

MST: Medium-sized telescope

MTBF: Mean time between failures

NAT: Network address translation

OES: Observation Execution System

OPC UA: Object Linking and Embedding for Process Control Unified Architecture

ORB: Object Request Broker

OTS: Off-the-shelf

PC: Pairwise comparison

QC: Product quality characteristics

RAM: Random Access Memory

RHEL: Red Hat Enterprise Linux

RPC: Remote procedure call

RTA: Real time analysis

RTT: Round-Trip-Time

SC: Strategic concerns

SL: Scientific Linux

SLICE: Specification Language for Ice

SST: Small-sized telescope

TAO: The ACE ORB

TCP/IP: Transmission Control Protocol/Internet Protocol

TCS: Telescope Control System

UI: User interface

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

INTRODUCTION

The ALMA Common Software (ACS) is a software framework for the construction of control systems. It was initially developed by the European Southern Observatory (ESO) for several years with the objective of unifying the software development of geographically distributed development teams of the ALMA Observatory. ACS tackles the challenge of constructing maintainable complex distributed systems. The first ACS prototype was presented in 2000 showing its core concepts [1].

Fortunately, ACS was intended to be a general framework from the beginning that could potentially be adopted by other organizations outside the ALMA collaboration. The LGPL license of ACS is an indicative of that. Over the years some organizations have adopted ACS for industrial and educational purposes [1].

In the recent years two projects, namely the LLAMA Observatory and the Cherenkov Telescope Array (CTA), have considered the use of the ACS framework to assist the development of their control systems. This interest has certainly revitalized the open source community behind the ACS project (ACS community) which continue to maintain and improve the framework to this day [1-3].

Despite the continuous efforts invested on improving ACS, this progress will be limited by some architectural decisions that were taken on the early 2000s that could have been favorable at that time, but now may be considered outdated (e.g., the use of CORBA implementations as the communication middleware).

The primary concern of this dissertation is the evaluation and selection of the communication middleware, which is a core component of many distributed systems. This selection has several architectural consequences that may limit the ability of ACS to fulfill the increasingly stringent requirements of modern observatories [4, 5].

Such decision cannot be taken arbitrarily due to the high risk of failure and the complexity of the problem. The organizations that support the ACS development often have to divide their resources between organizational-work and ACS development so it is unlikely that one of them have enough resources to follow a naive approach (e.g., integrating all the possible alternatives to ACS and then select the best) to solve this problem.

On the other hand, we cannot expect that interested parties will invest irrational amounts of resources in the evaluation. To solve this problem I propose an evaluation and selection process which objective is to reduce the risk of the system failing and its long-term cost. Furthermore the process can be tailored according to the resources available for the evaluation and the requirements of the organizations.

This dissertation has the following structure. Chapter 1 provides a more detailed discussion about about the problem we presented above. Even though this document is targeted for a highly specific audience (e.g., software engineers in the domain of astroinformatics) Chapter 2 provides a theoretical background with some basics concepts to understand the rest of this work. Chapter 3 describes the solution proposed (i.e., the evaluation and selection process) and the rationale supporting its construction. In Chapter 4, I execute the evaluation process, to the extent of my capability, applying it to the CTA system context with the objective of generating a guide that facilitates the full application of the process in the future and in doing so, validating the solution proposed. In Chapter 5 I provide some advices and recommendations that may help other evaluators during their execution of PECA. Finally Chapter 7 concludes the dissertation with some of the lessons learned during the evaluation process and provides insight about the future work.

CHAPTER 1: PROBLEM DEFINITION

ALMA (Atacama Large Millimeter/submillimeter Array) is a joint project between american, european and asian scientific organizations. The ALMA observatory combines 66 parabola antennas functioning as a huge radio telescope with a virtual diameter of approximately 16 km which are located in the middle of the Atacama desert, in Northern Chile, on the Chajnantor Plateau [6, p. 9].

Remote control of devices and basic components is a main requirement of the people involved in the ALMA project. Some of these components are related to the antennas such as the mount, unit control, correlator, thermostat, etc. Nowadays the software used by ALMA to control devices in a remote way is based on ACS (ALMA Common Software) framework which allows to standardize the software development between people that live in different parts of the world and have different development cultures [7].

At the same time, ACS is built on top of a CORBA compliant middleware which implements a standard for the Object Request Broker (ORB) technology. ACS relies on CORBA implementations to communicate components in distributed control systems using an object-oriented versions of remote procedure calls (RPCs) [8].

CORBA has been heavily criticized for having some serious deficiencies. Michi Henning, who worked as a chief CORBA scientist in IONA Technologies and then as a chief scientist in ZeroC between the years 2003 and 2010, in his famous article *The Rise and Fall of CORBA* [9] explains how the procedure followed by the OMG (Object Management Group) for defining the CORBA open standard was the main cause of its technical flaws which have been accumulating over time and eroded the interest in the use of this technology, due to political and economical factors. The main technical flaws mentioned in his article are the following: unnecessary complexity and inconsistency of the APIs, buggy and difficult to use C++ mapping and lack of key functionality such as support for security and versioning. More criticism can be found on the Internet within the reach of a Google search which includes bloated specifications and implementations, a steep learning curve, incompatibility issues with firewalls and NAT, high granularity and the need to recompile Interface Definition Language (IDL) after each change or use the even more complex dynamic interface. Other similar organizations such as the European Organization for Nuclear Research (CERN) are also interested in replacing CORBA in their control systems [10].

In fact, ACS is having problems in complying with the the QoS (quality-of-service) requirements of ALMA because it is based on CORBA. ALMA partially replaced CORBA with Distributed Data Service (DDS) technology for improving some ACS services such as the notification service [11], logging service [12] and bulk data transfer service [13]. Today there are some modern and mature technologies that might effectively replace CORBA in ACS entirely.

The goal of this dissertation goes beyond merely critcing CORBA. To the contrary, this work includes CORBA as a potential candidate to be evaluated with the new alternatives. We know that a lot of the problems that were criticized in the past were already solved by several CORBA implementations. If we can rescue something from these criticisms is that maybe the problems were solved too late because the base of CORBA developers is shrinking to inconvenient levels. Finding CORBA developers in the future may be impossible or very expensive as CORBA is mostly considered a niche technology.

We also should consider that CORBA was conceived about 27 years ago as an answer to an interoperability problem in very heterogeneous distributed systems in terms of hardware (e.g., embedded systems), software (i.e., different operating systems) and programming languages. We can argue that this great interoperability support is a big contributor to the complexity of CORBA. However, not all modern observatories are in need of such interoperability. For example the Cherenkov Telescope Array (CTA), which is the test subject for this document, forced an homogeneous system at least in terms of hardware and software for running the Array Control and Data Acquisition (ACTL) software which is based on ACS, with OPC UA ensuring interoperability with the devices. This means that they have to pay the full price of CORBA interoperability (e.g., lower performance [14]), but they are not going to profit from it. Moreover, maybe they can’t even pay the price at all because the manpower for developing and maintaining ACTL is very limited [15] and the high complexity of the CORBA APIs plus reduced availability of CORBA developers might be a showstopper for the ACTL development team. So it is not surprise that CTA adopted ACS with suspicion due to its communication middleware. It becomes necessary to study the different alternatives that didn’t exist when CORBA was chosen as the base middleware for the construction of ACS.

With the explosive growth of open source software, developers often have to make a decision between building software from scratch or reusing software off-the-shelf (OTS). Moreover, the great availability of high quality open source projects makes it difficult to choose the software that best fits their needs and often a naive approach is used for the selection. The middleware software is not an exception. Even though we can use a naive selection method for trivial software selection [16], it is not an appropriate approach for communication middleware selection in ACS due to the high cost of replacing such a core component and the high risk of not meeting the requirement specifications. When software developers or architects choose a specific communication middleware, several design and architecture decisions about the whole distributed system are being taken under the hood that have very important short and long-term consequences, including the success or failure of the project.

1.1. Objectives

The objective of this dissertation is to create a guide that serves as support for the evaluation and selection of communication middleware, with the goal of replacing the CORBA middleware in ACS with a modern technology aligned with the current and emerging requirements of modern observatories such as CTA.

1.1.1 Specific objectives

- To analyze the relevant evaluation criteria for the modern observatories.
- To analyze the available technologies for replacing CORBA in ACS.
- To show a partial execution of a reusable approach for the evaluation and selection of communication middleware.

CHAPTER 2: BACKGROUND

2.1 ALMA Common Software

ALMA Common Software (ACS) is a software framework for the development of distributed control systems. It was initially created to provide a common platform for unifying the software development of all partners in the ALMA collaboration [7, p. 1]. It supports the development of maintainable systems while trying to hide the complexity of the underlying tools that compose it (e.g., CORBA middleware) with the implementation of a Container Component Model (CCM) [7].

ACS provides some communication protocol implementations, based on CORBA middleware, which enable the interaction between remote components (developed in C++, Java and Python). These components are often used to control hardware devices. It also provides several common and centralized services which are frequently needed in distributed control systems such as logging, alarms, error handling, configuration database, archive and object location [7, p. 3].

Even though ACS was designed for its use in ALMA (an astronomical interferometer of radio telescopes on northern Chile), it was released under a LGPL license which has motivated several organizations around the world to use it in industrial and educational applications [1]. For example the Italian National Institute of Astrophysics (INAF) uses ACS for the development of their own control system named DISCOS which is used in three radio telescope facilities [17].

ALMA is not currently developing new features for ACS because the project is in production phase [5]. An ACS open source community, composed primarily by some organizations interested in ACS, ALMA engineers (volunteers) and university students, is now in charge of maintaining and extending the ACS framework with new features for projects that choose to incorporate ACS in their software development without the restrictions of the ALMA project [1]. In the recent years some organizations such as the LLAMA Observatory and the Cherenkov Telescope Array have decided to incorporate the use of ACS in their control systems. This renewed interest and the new resources have strengthened the ACS community, but at the same time it has set up the challenge to make ACS a framework able to support the demanding requirements of modern observatories.

2.2 Cherenkov Telescope Array

An international partnership of scientists from 32 countries is developing the Cherenkov Telescope Array (CTA). This next-generation gamma ray observatory will address complex scientific topics such as “the origin and role of relativistic cosmic particles, probing extreme environments (i.e., neutron stars, black holes and cosmic voids) and exploring frontiers in physics (i.e., dark matter, photon propagation and axion-like particles)” [18, p.

3]. CTA will deploy telescopes in two sites, one on each hemisphere, allowing a full sky coverage and improved sensitivity compared to existing observatories [18].

CTA will be composed by at least three types of telescopes that allow the scientists to capture data about gamma rays with different levels of energy levels. The low energy gamma rays are captured by the large-sized telescopes (LTSs). The gamma rays in the core energy range are captured by the medium-sized telescopes (MTSs). Finally the high energy gamma rays are captured by the small-sized telescopes (STSs) [18].

The southern array will be located in the European Southern Observatory Paranal site in Chile and will include 99 telescopes (4 LTSs, 25 MTSs and 70 LTSs). The northern array will be located at the Instituto de Astrofisica de Canarias (IAC) Roque de los Muchachos Observatory in La Palma, Spain and will be equipped with 19 telescopes (4 LTSs and 15 MTSs) [18].

The CTA software engineers have to develop low-level software to access the telescope assemblies and auxiliary devices. Then this low-level software has to be integrated with a high-level array control system, leading to a highly complex distributed system.

This high-level system, known as the Array Control and Data Acquisition (ACTL) software system is responsible for orchestrating and executing the observations of CTA. To accomplish this the ACTL system has to provide several functionalities such as “monitor and control all telescopes and auxiliary devices in the CTA arrays, schedule and perform observations and calibration procedures, and to time-stamp, read-out, filter and store data” with some strict non-functional requirements [19, pp. 2-4]. The ACTL software development is based on the ACS framework.

The low level software is developed by the telescope and device teams. It is based on the Object Linking and Embedding for Process Control Unified Architecture (OPC UA) technology used to standardize the hardware access. The *bridge* design pattern is used to integrate the low-level software with the ACTL software [15].

End-to-end prototypes for some telescopes subtypes are ready. The SST-1M prototype provides an ACS component for each subsystem of the telescope. ACTL will be able to control the SST-1M telescope by issuing commands to the telescope's ACS master component which can connect to and control each subsystem's components. Finally these components access the hardware through OPC UA to execute the commands [20]. In an analogous way the ASTRI SST-2M prototype provide the Mini-Array Software System (MASS) to control telescopes. The ASTRI team built MASS in a modular way that allows an easy integration of the ASTRI SST-2M telescopes with ACTL high level software by simply defining an interface to the the Telescope Control System (TCS) component [21].

In 2017 CTA internally changed the name of the ACTL system to Observation Execution System (OES). The majority of the documentation that was analysed during the development of this dissertation uses the former, so *ACTL* is utilized in this document. Nevertheless, they may be considered interchangeable in this dissertation.

2.3 Communication middleware

As described in [22] distributing a software application almost always adds complexity, concurrency issues and performance degradation, so it should be done only after a careful evaluation. Usually we build a distributed system when we are addressing an inherently distributed problem (e.g., aircraft flight control) and/or we need to boost certain properties (e.g., fault tolerance and scalability). For both of these reasons, as we briefly explained in Section 2.2, the ACTL team can not develop the CTA software system following a monolithic approach.

The software components that interact in a distributed system are not being necessarily executed in the same process or even the same hardware so they need a mean to communicate. This can be done by using the network layer protocols like using TCP/IP sockets directly. The problem is that working in such low level has several disadvantages (e.g., it is error prone, difficult to scale and hard to maintain) [22, 23].

To overcome these disadvantages, we might use an extra layer of software between the application and the network services provided by the operating system kernel [22, 23]. This layer, known as *communication middleware*, allows the developers to work at a higher level of abstraction. This technology may bring several advantages (and disadvantages) to the system, depending on the protocols and services it provides and the quality of their implementation.

A communication middleware often implements one or more basic communication styles such as remote procedure call (RPC), message-oriented communication, shared memory and/or streaming-oriented communication [22, 24]. Other more complex communication styles and paradigms are built on top of these basic styles. There are several ways to classify communication middleware. Alrahmawy [25, Ch. 2] presents a taxonomy of middleware paradigms:

1. The Message Passing Paradigm
2. The Client Server Paradigm
3. The Peer-to-Peer Paradigm
4. The Message System Paradigm
    a. The Point-To-Point Message Model
    b. The Publish/Subscribe Message Model
5. The Remote Procedure Call
6. The Distributed Object Paradigm
    a. Remote Method Invocation
    b. The Network Service Paradigm
    c. The Object Request Broker Paradigm (ORB)
7. Component Oriented Architecture (COA) Paradigm
8. The Application Server Paradigm
9. The Tuple Space Paradigm
10. The Collaborative Application (Groupware) Paradigm

After careful consideration, I decided to use this classification due its completeness (e.g., it includes 10 paradigms). Furthermore it is ordered by level of abstraction (starting from the lowest level) which is an important attribute to consider in the evaluation of communication middleware.

There is no clear distinction between the work that should be done at the application layer and the middleware layer. Generally, low level middleware just provides one or more communication protocol implementations while high level middleware also provides common, reusable and domain-independent services and well tested patterns that allow the developers to focus more on the business logic of the application.

ACS was built on top of CORBA standard compliant communication middleware: The ACE ORB (TAO) [26], JacORB [27] and OmniORB [28]. These ORBs provide the core communication capabilities for the ACS components which are used to build distributed control systems. Also, each of these implementations of CORBA standard provide its own set of common services. ACS services are based on some of these CORBA compliant services. Even though the vanilla CORBA middleware can be classified under the Object Request Broker Paradigm (see Section 2.3), ACS uses CORBA to implement a custom Component Container Model (CCM) [7] which allows the ACS users to work at an even higher level of abstraction (Component Oriented Architecture Paradigm).

2.4 Component Based Development

The increasing availability of reusable software promoted the arising of a new software development paradigm, in which the focus has changed from programing complex software systems from scratch into composing software systems like a puzzle. This idea has incarnated in the already well-known Component Based Development (CBD) process in which the systems are composed by in-house and third-party reusable software [29].

We can see how this vision of software development is present in ACS and ACTL. On one hand the ACS framework reuses third-party software such as the communication middleware (more details in Section 4.1.4). On the other hand the ACTL reuses software frameworks (e.g., ACS and OPC UA). This is not a surprise since the total estimated cost of the ACTL project is around 18 million euros from which the labour and software-development cost is the dominating factor (13 million euros equivalent to 165 FTE) [15]. CBD, if executed properly, reduces de development cost (i.e., time and money) and it might even boost the overall quality of the system. Nonetheless CBD puts new challenges on the table: The selection of software components and their integration in a software system that has to comply with its requirement specification and the system context constraints.

One of the major risks of CBD is choosing the right components. Extensive work has been done regarding the evaluation and selection of commercial Off-The-Shelf (COTS) component. Comella-Dorda et al. [16, Sec. 1.1] describe the characteristics of COTS products:

- Sold, leased or licenced to the general public
- Offered by a vendor trying to profit from it
- Supported and evolved by the vendor, who retains the intellectual property rights
- Available in multiple, identical copies
- Used without modifications of the internals

2.5 Evaluation and selection of COTS

Garg [30] made an extensive list of the COTS evaluation processes designed until January 2017. In his work he pointed out the general advantages and disadvantages of using each method. He concludes that no process is inherently better than the others.

Bali and Madan [31] presented a literature review of the empirical studies carried out in the past for evaluation and selection of components during the design phase of Component Based Software Systems (CBSS). Feras Tarawneh et al. [32] presented a state of the art about evaluation and selection of COTS, exposing the common strategies and the issues that the methods should address. Additional common limitations of the evaluation processes are described in [33].

A specially through analysis was performed by Mohamed, Ruhe and Eberlein [34] which includes a comparison of the eighteen more relevant selection process to that date (March 2007), stating their contributions to COTS evaluation in historical order and finally stating the pros and cons of each method.

From these studies I compiled a list of relevant criteria that we should consider when choosing a process for the evaluation and selection of communication middleware for ACS. These criteria are described below.

2.5.1 Qualities of COTS selection approaches

This section describes six features from the evaluation and selection process that are desirable.

Conformance to the GCS Method

The General COTS Selection Method (GCS) represents a series of steps that are commonly followed by the majority of the COTS Selection processes even though there is not an universally accepted method. The steps of the GCS Method are the following [34, Sec. 2.1]:

- Step 1: Define the evaluation criteria based on stakeholders requirements and constraints.
- Step 2: Search for COTS Products.
- Step 3: Filter the search results based on a set of ‘must have’ requirements. This results in defining a short list of most promising COTS candidates which are to be evaluated in more detail.
- Step 4: Evaluate COTS candidates on the short list.

- Step 5: Analyze the evaluation data (i.e., the output of Step 4) and select the COTS product that has the best fitness with the criteria. Usually, decision making techniques, e.g.analytic hierarchy process (AHP), are used for making the selection.

Tailorability

As described in [34, Sec 3.1] tailorability refers to “the flexibility of the evaluation process (including the evaluation criteria themselves) to be adapted based on the available effort for the project.” This is a rare quality of the COTS selection approaches.

Compatibility with common evaluation strategies

Even though the evaluation processes might differ in the details of their implementation, they commonly rely on some of the three general strategies to implement the evaluating step. These strategies are the following [34, p. 2]:

- Progressive Filtering: It attempts to reduce the number of potential candidates by executing several iterations of the evaluation process. In each iteration the evaluators use discriminating criteria to reject unsuitable alternatives.
- Puzzle assembly: Sometimes the fitness of COTS software depends on its interaction with other products. In these cases the requirements of both the COTS software and the interacting products should be considered simultaneously.
- Keystone identification: The evaluators identify a key requirement and then they evaluate if the candidates can fulfill them, effectively filtering a lot of unsuitable candidates in the process.

Suitability for single and multiple COTS selection

The vast majority of approaches for COTS selection were designed for single COTS selection, that is, they are used for selecting one product at a time. There are a few processes that allow to evaluate and select a combination of products in a parallel fashion. Nonetheless, the approaches for multiple COTS selection are more expensive in the short term.

Mismatches handling support

When we develop Component Based Software System (CBSS) two types of mismatches might occur when selecting a COTS software. Tarawneh et al. [32, Sec 2.1.1] describe the architectural mismatches as the discrepancies between the COTS and the rest of the system due to interoperability issues and incompatibilities such as “using different database schema, different programming languages, or communication protocols” [32, Sec. 2.1.2]. They also describe the COTS software mismatches as the differences between COTS software capabilities and user requirements or expectations. After identifying the mismatches, the evaluation team should also be able to provide a way to fix them, the resources needed and the risk associated to the proposed solutions.

Availability of tools

The final criterion presented in [34] is the availability of tools to facilitate the application of the selection process. Some approaches are supported by software tools which are primarily prototypes developed by the authors.

2.5.2 Issues of COTS selection approaches

There are common issues in COTS selection that are not addressed by the previous criteria and are considered undesirable. Some of these issues are mentioned in [32] and other ones are mentioned in [33]. This section describes six of these issues.

Having a single evaluation criteria

Approaches based in only one criterion could reach a deep level of analysis about a specific aspect of the software, but the evaluators would be (probably) ignoring several critical factors [33].

Non-functional requirement problem

Nowadays the importance of non-functional requirements (NFRs) in software development is well understood. The NFRs describe how the software system will do its work and thus they often have a great impact on the system architecture. Some selection methods don’t deal with these requirements or have poor handling of them [33, 32].

Lack of learning from previous COTS software selections

Software components chosen in the past, successful criteria and techniques used in previous evaluations and data collected about vendors and open source communities may be very useful during COTS software selection processes [32]. For these reason many selection methods promote the documentation of the COTS software evaluation, but they “don’t show mechanism to store and manage the information” [32, Sec. 2.3].

Assuming user requirement exist

“Without well understood and properly obtained user requirements it is not possible for the evaluators to produce quality results” [33, Sec. 3]. Many approaches assume that these requirements were already elicited and don’t support this task [34].

Higher complexity/effort level

Cost is a sensitive topic in COTS evaluation because an important goal of applying a COTS selection method is to reduce long-term cost of choosing an inappropriate product, without paying too big a short-term cost. As pointed out in [33] evaluators are not inclined to follow effort-intensive and time-consuming selection approaches.

Not describing in detail, What to do?

One important criticism to selection processes is the lack of detailed specification of what to do and how to do it. This is arguably one of the main reasons that motivates evaluators to follow an ad-hoc approach [33].

2.5.3 Peca evaluation process

Software Engineering Institute (SEI) and National Research Council Canada (NRC) created the PECA evaluation process as an alternative to evaluate COTS (Commercial off the shelf) software in a comprehensive, continuous and defined process that consider context, uncertainty and is based on facts [16].

“PECA is named by the four activities that make up the process: Planning the evaluation, establishing the criteria, collecting the data and analyzing the data” [16, p. 11]. This evaluation process is highly tailorable and allows to perform a formal evaluation of open source technologies with virtually no modifications.

The input of the process are the list of possible candidates and the system requirements. Sometimes the stakeholder expectations are not fully considered in the system requirements, but they could potentially (and probably) influence the evaluation. Another input is the evaluation guide which contain the *know how* of the organization about processes and techniques for performing evaluations [16].

The output of the process is an improved evaluation guide, a product dossier (repository of software documentation), an evaluation record that contains a description of the evaluation process and a summary/recommendation that includes the results. It is also worth noting that PECA has favorable side effects. Even if the evaluation fails and no suitable candidate can be chosen, this process improves the insight of the evaluators about the system. Engineers might use this knowledge to refine the system requirements and also architects and integrators get feedback [16, pp. 13-14].

While PECA steps are a high-level description of what evaluators have to do, evaluation techniques used in each step are low level descriptions of how it should be done. Appendix A includes a brief summary of the process as described by Comella-Dorda et al. in [16].

2.6 Techniques for evaluating software components

2.6.1 Hierarchical Decomposition Method

Similar to the well-known Goal-Question-Metric approach, the Hierarchical Decomposition Method is a technique used in multiple-criteria decision-making (MCDM) problem solving. This dissertation focuses on the implementation proposed by Kontio, Caldiera and Basili [35] which is tailored for supporting OTS software evaluation. This hierarchical decomposition method breaks down a problem into evaluation goals, evaluation criteria and finally in measurable evaluation attributes. The result is a tree representation of the problem.

This method enables the evaluation team to start working from a high level description of what is important (factors) to meaningful and measurable criteria (evaluation attributes) as illustrated in Figure 2.1.

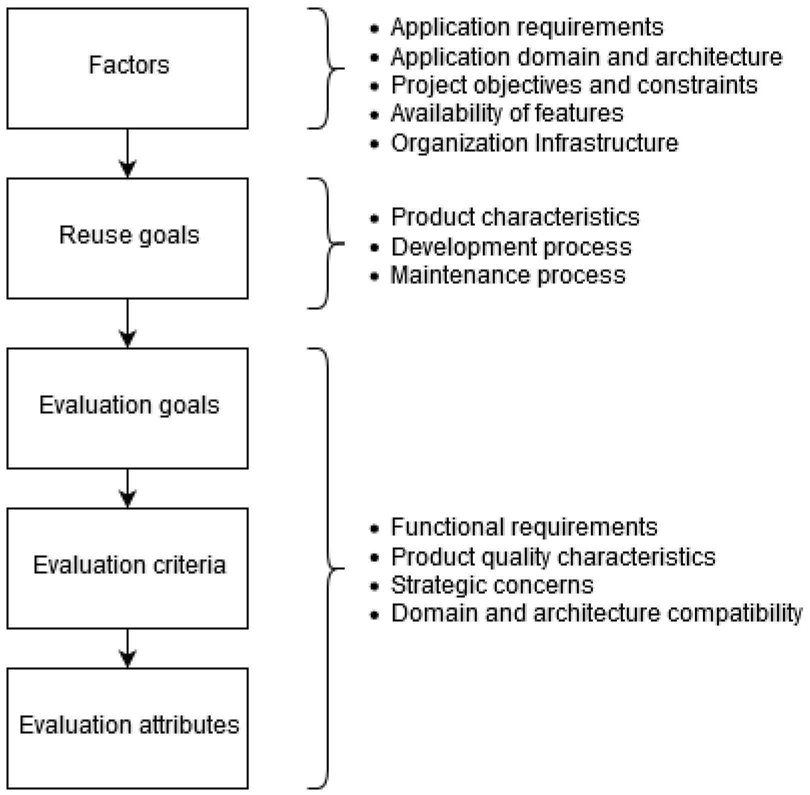

Factors
Reuse goals
Evaluation goals
Evaluation criteria
Evaluation attributes
• Application requirements
• Application domain and architecture
• Project objectives and constraints
• Availability of features
• Organization Infrastructure
• Product characteristics
• Development process
• Maintenance process
• Functional requirements
• Product quality characteristics
• Strategic concerns
• Domain and architecture compatibility

Figure 2.1: Relationship between the key concepts in the Hierarchical Decomposition Method.
Source: Elaborated by the author.

The steps of the hierarchical decomposition method presented in [35] are summarized as follows:

- Analyse the influencing factors: these factors come from five sources: the application requirements, the application domain and architecture, the project objectives and constraints, the availability of features and the organization infrastructure.
- Identify and state the reuse goals: from the analysis of the influencing factors we can derive the reuse goals of the system. These goals represent what is expected from the reuse of the off-the-shelf software (e.g., the communication middleware). These expectation might be about the product characteristics or about the impact of the software in the development and maintenance process. A reuse goal statement should be documented although it will probably be abstract and simple at this point.
- Identify and formulate evaluation goals: the evaluators can easily determine the goals of the evaluation from the reuse goals. These goals are documented using the Goal-Question-Metric notation.
- Define high level criteria: for each evaluation goal, the evaluation team has to "define a set of high level criteria or questions that characterize it" [35, p. 6]. Each criterion can be classified as a functional requirement, a product quality

characteristic, a strategic concern or a constraint for achieving domain and architecture compatibility. For each criterion, the evaluation team has to "write down an unambiguous definition of it" [35, p. 6].

- Identify the evaluation attributes: the evaluation team has to mark a criterion as an evaluation attribute if "the value for the criterion can be determined with an objective measurement, observation or judgment" [35, p. 6]. If it is not the case, then the evaluators have to continue decomposing it. These evaluation attributes are the leaf nodes in the hierarchical representation of the MCDM problem.

2.6.2 Analytic Hierarchy Process (Pairwise Comparisons and Sensitivity analysis)

The Analytic Hierarchy Process (AHP) is a framework of logic and problem-solving [36] developed by Thomas Saaty for MCDM which is widely used across several different domains. Saaty argues that AHP "is based on the innate human ability to use information and experience to estimate relative magnitudes through paired comparisons" [36, Sec. 1.1].

Brunelli [37, Fig. 1.1] positions AHP in the intersection of decision analysis and hard operations research. He also explains that AHP really excels in MCDM problems where we have to deal with tangible criteria (i.e., objectively and unambiguously measurable) in pair with intangible criteria which have not a standard scale and thus they are hard to measure (e.g., vendor reputation, open-source project quality and extensibility support). AHP address this issue using pairwise comparisons which results are compatible with the theory of relative measurement [37, p. 6].

Saaty [38, p. 3] summarized the decision making process in the following steps:

- Structure the problem with a model that shows the problem's key elements and their relationship (e.g., hierarchy).
- Elicit judgements that reflect knowledge, feelings and emotions.
- Represent those judgments with meaningful numbers.
- Use these numbers to calculate the priorities of elements of the hierarchy.
- Synthesize these results to determine an overall outcome.
- Analyze the sensitivity to changes in judgements.

We are only interested in the techniques of AHP that we might import to PECA such as the pairwise comparisons (PCs) and weighted aggregation. They allow us to prioritize the evaluation criteria and consolidate heterogeneous data collected during the evaluation process. The sensitivity analysis is also an useful technique for analyzing the consolidated data in PECA.

2.6.3 Quick Assessment (Filtering)

PECA suggest the use of filters as an effective way to narrow a big list of candidates by discarding those who are clearly unsuitable due to not meeting certain easily measurable criteria. This is very useful because measuring and collecting data can be an expensive process and filtering helps to make the process more cost-efficient by enabling the evaluators to focus only in the most promising candidates.

Filtering is an intuitive task in which the evaluators select some easily measurable criteria as showstoppers to filter the whole pool of candidates available. Nevertheless Wasserman, Pal and Chan [39, pp. 11-12] propose some useful guidelines and good practices for the initial filtering and quick assessment of open source software. They describe five steps:

- Decide the target usage of the application: they identify four general types of usage (mission-critical, regular, development and experimentation).
- Select a handful of viability indicators based on target usage: these are quick and easy to use indicators that strongly show the viability of the candidates. In [39] a list of common indicators for filtering open source software is provided.
- Add more internal viability indicators to the list if applicable: the candidates may need to meet some crucial requirements coming from the target software system or its context. The evaluation team should include these indicators if they are relevant and easy to measure.
- Create a policy of passing criteria: the evaluators have to define a policy to identify what assessment outcomes are appropriate. Easily measurable indicators often provision a positive or negative outcome. Sometimes evaluators might introduce an intermediate category. For example, an acceptable policy might be *no more than two negatives*.
- Assess each software component against the list of viability indicators: the evaluation team can start the quick assessment after they defined the viability indicators and the passing policy. The result will be a reduced list of candidates which are evaluated in more detail in the next steps of PECA.

CHAPTER 3: SOLUTION

3.1 Choosing the approach

As stated in Section 2.5 there are several approaches to evaluate and select COTS software in a methodical way. Furthermore there is not a silver bullet in the field of COTS evaluation and selection. How we choose the best approach to our context? In this section the rationale behind the choice of PECA for this work and the CTA middleware selection is summarized: Table 3.1 describes how PECA stands against the criteria presented in Section 2.5.1 and Table 3.2 state how PECA overcomes the common issues in COTS selection introduced in Section 2.5.2.

Table 3.1: Qualities of PECA as an OTS software selection approach.
Source: Elaborated by the author

Quality	Description
Conformance to the GCS Method	PECA contains all the steps of the GCS Method. Additionally, it proposes a planning step at the beginning of the process.
Tailorability	Tailorability is one of the most relevant factors to choose PECA for this evaluation. Only two approaches for COTS selection are considered tailorable and according to [34], PECA is the most detailed and it provides some guidelines for experts to tailor the process. Moreover the tailorability of PECA makes it useful in many contexts. As mentioned in [16] PECA is equally applicable to COTS-like software (i.e., that doesn't comply all the conditions stated in Section 2.4 for COTS products) including open source software which is the focus of our evaluation.
Compatibility with common evaluation strategies	Typically the COTS selection methods imply the use of one or two of the common strategies. PECA is flexible enough to allow the use of any combination of these strategies so the evaluators are able to choose the best approach to fulfill their goals under their current constraints.
Suitability for single and multiple COTS selection	PECA doesn't support multiple COTS selection, but I consider that this issue is not relevant in our context. We are interested in evaluating the communication middleware that should be used in the ACS framework. A combination of technologies that covers all the requirements of the middleware might be technically optimal, but it will significantly increase the development, integration and maintenance of a more complex system. Furthermore, the execution of multiple COTS selection approaches require the allocation of a massive amount of resources [40] which is not desirable, even for CTA.
Mismatches handling support	Full support for mismatches handling is very rare and expensive in the short term [34]. PECA is one of the few approaches that provide a partial support in a cost-effective way, but it relies on expert knowledge to do it [16]. This should not be a problem for CTA, but it might be a showstopper for other organizations with less available resources.
Availability of tools	As a tailorable process, PECA might use tools designed to support the techniques selected for implementing the process. For example PECA can use PCs, so a tool such as Super Decisions v3 might be used to support this technique as is shown in Chapter 4.

Table 3.2: PECA versus the common issues in COTS selection approaches.
Source: Elaborated by the author

Issue	Description
Having a single evaluation criteria	PECA promotes the inclusion of multiple criteria addressing different aspects of the OTS software product and the target system.
Non-functional requirement problem	PECA offers good coverage for these requirements. In the *Establishing the criteria* step, the method instructs to consider all the relevant sources of NFRs for defining the evaluation requirements such as architecture/interface constraints, programmatic constraints, operational environment and stakeholder expectations.
Lack of learning from previous COTS software selections	PECA provides guidelines and templates to help the evaluators to create a knowledge repository. This repository is intended to serve as an input (and output) in each iteration of the process to improve the reuse maturity of the organization.
Assuming user requirement exist	Although system requirements are an important input for PECA, the process is not always executed sequentially. “Evaluation events, such as a need for new criteria to distinguish products or unexpected discoveries while collecting data can lead to the start of a new iteration” [16, Fig. 4] so we can still work with a partial definition of the requirements, but it is not recommended.
Higher complexity/ effort level	PECA was designed with cost-efficiency in mind. Moreover, its tailorability allows it to adapt to different organizations, COTS-based development processes and availability of resources [16].
Not describing in detail, What to do?	Even though PECA is characterized as a “detailed tailorable COTS selection process,” it relies on “human experience” to tailor the process [34, Table 12] and it doesn’t detail how to execute the different techniques required during the process. With this dissertation I attempt to overcome this issue by providing a concrete execution of the approach for middleware selection.

3.2 Implementing the evaluation process

PECA provides high level guidelines about what evaluators should do, but also provides enough flexibility to let them choose how to do it. Each step of PECA suggest a series of tasks that has to be completed (See Appendix A). Some of these tasks are straightforward so they don’t require additional low level guidelines. Nevertheless there are some more complex tasks that require a more detailed explanation on how to do them so evaluators can execute the process effectively and painless. After selecting PECA as the evaluation and selection process, the evaluators should tailor it to the current system context and one of the ways to do it is to select appropriate techniques that define the low level implementation of the process. For this dissertation I purposely chose techniques with low cost for the evaluators so they can be executed by me and also they can be attractive for other evaluators which often reject the use of expensive techniques in the context of software selection [33].

3.2.1 Hierarchical Decomposition

Contrary to what some overconfident reader might think, defining evaluation criteria is far from being a trivial task. Hierarchical Decomposition and the Goal-Question-Metric approach are two alternatives recommended by PECA for accomplishing that. Nevertheless these are generic techniques which can be used in a wide variety of dominions. There is a lack of explicit documentation about the application of these techniques for software product evaluation and adapting them to this domain may be time consuming and error prone.

Fortunately in [35] a hierarchical decomposition method is presented which claims to be effective, having low overhead and most importantly being publicly and detailed explained in the context of software product evaluation (Off-The-Shelf software evaluation).

As described in Section 2.6.1 the hierarchical decomposition method allow us to generate a tree representation of the decision problem (i.e., What middleware should we use?). Therefore this method covers the implementation of the first two task of the *Establishing the criteria* step of PECA: defining the evaluation requirements and defining the evaluation criteria (See Appendix A).

The method follows the general guidelines of PECA so we can easily incorporate this technique in our process. It is also important to notice that the terminology of the hierarchical decomposition method maps naturally with the terminology used in PECA. The Table 3.3 shows this mapping.

Table 3.3: Mapping between PECA and the hierarchical decomposition method terminology.
Source: Elaborated by the author.

PECA process	Hierarchical decomposition method
Source of evaluation requirements (e.g., system requirements).	Factors
Evaluation requirements (Goal)	Reuse goals
-	Evaluation goals
Capability Statements (Question)	Evaluation criteria
Measurement Method (Metric)	Evaluation Attribute

3.2.2 Analytic Hierarchy Process

AHP (specifically the pairwise comparison technique) is recommended by PECA [16] and by the hierarchical decomposition method [35] to prioritize the evaluation criteria in the *Establishing the criteria* step. Moreover we might also use pairwise comparisons for the consolidation of heterogeneous data in the *Analysing the data* step. AHP also describe how to make a sensitivity analysis which is an useful technique for analyzing the data after they have being consolidated.

AHP has generated lot of controversy [41, 42]. It has been widely adopted and used successfully by many practitioners, and yet it also has been heavily criticised by some theoretical academics. Emrouznejada and Marra [43] attempt to provide an objective literature review about the state of the art of AHP from 1979 to 2017. In this review they describe some advantages and criticism associated with AHP which are summarized below.

On one hand AHP is easy to use and flexible. It allows its users to tackle complex MCDM problems by guiding the construction of a hierarchical model of the problem. It has being applied in several different domains that range from health and education to computer science applications. Moreover AHP allows the users to emit verbal judgments about the priorities of the criteria and the fitness of the alternatives. They also can check the consistency of their judgments. Finally AHP assist the users to deal with the potential bias in group-decision making [43].

On the other hand, some critics has questioned the capability of the “principal right eigenvector of the pairwise comparison matrix to produce true rankings” [43, Sec. 5.2]. Other criticism is related to the potential rank reversals when a non-optimal alternative is introduced. In addition AHP relies heavily on the consistency of the user’s judgment and reaching consensus in group-decision making may be difficult [43, Sec. 5.2].

Saaty [36] mention several reasons that promoted the widespread use of AHP in academia and organizations. Here are the main reasons for using AHP in this dissertation and even by organizations that are on a similar situation as CTA:

- Its simplicity make it suitable for evaluations teams which members have different technical backgrounds (e.g., software developers, operators and astronomers). Moreover people that never used AHP do not require much effort to master it, making it a low-cost alternative for our evaluation.
- Even though AHP is simple, it allows to address complex MCDM problems and the middleware selection for ACS clearly falls under this category. One of the more notable strengths of AHP is its ability to consider intangible criteria which are difficult to quantify (e.g., coupling) at the same time as quantifiable criteria (e.g., bandwidth usage) which often have standard ratio scales or even an absolute scale.
- It can be executed by a single evaluator or by an evaluation team. In case of group-decision making it promotes compromise and consensus. It also allows to negotiate the conflicting interests of the stakeholders.
- AHP doesn’t impose any restriction in the size of the hierarchy used to represent the problem. If evaluators want to reduce the effort required in the evaluation they may use a simpler hierarchy than, for example, the one we present in Section 4.2.5. Nevertheless simplifying the model increase the risk of obtaining irrelevant results.

3.2.3 Other techniques

PECA [16] suggest the use of some techniques which importance is self evident, such as the gap analysis and the cost of fulfillment. In the next chapter the reader can see that I generated a hierarchy focused on the benefits provided by the middleware to the system. Because we are dealing with open source software the cost of choosing one alternative can't be trivially calculated. Using gap analysis and cost of fulfillment, the evaluation team can also provide information about the cost of choosing each alternative to the decision makers.

3.3 Summary of the solution

Table 3.4 shows the proposed evaluation and selection process and its low-level implementation.

Table 3.4: Summary of the evaluation process and my implementation proposal.
Source: Elaborated by the author.

PECA (high level)		**techniques (low level)**
Planning the evaluation		Straightforward
Establishing the criteria	Defining the evaluation requirements	Hierarchical decomposition method
	Defining the evaluation criteria	Hierarchical decomposition method
	Prioritizing the criteria	Pairwise Comparisons (AHP)
Collecting the data		Quick assessment, literature reviews and hands-on techniques
Analysing the data	Consolidating the data	Pairwise comparisons and weighted aggregation (AHP)
	Analysis	Sensitivity analysis (AHP), gap analysis and cost of fulfillment
	Making recommendations	Straightforward

CHAPTER 4: SOLUTION VALIDATION (THE GUIDE)

The purpose of this chapter is twofold: it validates the proposed middleware selection process (See Section 3.3) based on [16] by showing its application on a real case (CTA) and together with Chapter 5, it constitutes a guide that other evaluators may use if they are interested in implementing the process themselves.

One of the main issues of COTS selection processes is that they don't always specify how to do it. In this chapter we can see how to implement the proposed solution for selecting the best communication middleware for the ACS framework which has to satisfy the needs of the ACTL software system. Due to the fact that the execution of the evaluation process for this particular case requires too much work for a lonely evaluator I only provided a partial execution. Nevertheless I covered the whole process end-to-end in a way that allows reader to understand the details of the implementation and apply the same principles while doing a complete execution.

4.1 Planning the evaluation

Even though this step is relatively straightforward, it is very important because it allow us to unambiguously define the scope of the evaluation. For exemplification, in this section I show the outputs of each task generated during my PECA iteration.

4.1.1 Forming the evaluation team

The first task is to form the evaluation team. In this execution of PECA I am the only member of evaluation team (see Table 4.1). Recommendations for assembling a good evaluation team are presented in Section 5.1.

Table 4.1: Evaluation team.
Source: Elaborated by the author

Name	Renato Fulvio Sanhueza Ulsen
Description	Thesis student of computing engineering from the Federico Santa María Technical University which is located in Valparaíso, Chile.
Experience	Six months of experience working as a software architect and developer in Lifeware SAS.

4.1.2 Creating the charter

The evaluation charter for this iteration is presented in Table 4.2. PECA requires commitment from both the evaluators and the decision makers [16]. Due to the fact that this is an external evaluation meant to serve as an example, the evaluation team was unable to obtain commitment from the relevant decision makers, but it will be mandatory for future iterations of PECA.

Table 4.2: Evaluation charter.
Source: Adapted from [16]

Statement of evaluation goals	Generate a guide for the evaluation and selection of the communication middleware of ACS so it can support the software development of modern observatories such as CTA.
Scope of evaluation	First a set of technologies selected by the evaluation team are filtered. Then the candidates that pass the initial filter will be evaluated deeper.
Team member and roles	I am the only evaluator. There are not decision makers in this first iteration.
Explicit state of commitment	Doesn't apply in this iteration.
Summary of factors that limit selection	Time frame = 16 weeks.
Summary of decisions that have already been made	The guide will show an end-to-end execution of the process (i.e., all the steps and tasks of the process will be executed). Nevertheless, due to budget constraints, it will be a partial execution (i.e., the evaluation team will only collect and analyze the data about two criteria for showcasing the process). The communication middleware must be open source software.

4.1.3 Identifying stakeholders

There are at least four types of stakeholders that are relevant to this evaluation:

- Software architects: they are interested in the architectural consequences of choosing a communication middleware and how they affect the lifecycle of the system.
- Software developers: they care about how the communication middleware will impact the software development process of the system.
- Operators: they care about the usability and maintainability of the system.
- Astronomers: they need that the system implements the functional and non-functional requirements so they can get the desired outcome of its operation (e.g., meaningful observation results).

4.1.4 Picking the approach

Evaluation depth

The depth of the evaluation depends on two factors:

- Risk of failure: the measure of the impact on the system if a wrong product is selected [16]. I estimated the risk by analyzing the architecture of the current ALMA ACS implementation described in [8]: ACS is a software framework located between the application layer (e.g., applications, COTS and shared software) and the operating system. It serves as a glue between the components of the distributed control system of ALMA. It is based on CORBA middleware which provides communication facilities. Figure 4.1 shows an UML Package Diagram of ACS. The packages are grouped into four layers. Each package can only use packages on the same layer or the lower layers.

We can see that the CORBA middleware is in the lowest layer. Many packages on the upper layers depend on the CORBA middleware and so do the applications developed by ALMA. It is evident that the risk of choosing the wrong technology to replace CORBA is too high, because it requires the modification of a significant amount of software modules, services and applications developed on top of the middleware, with the financial cost that that implies (e.g., training, design and development effort). Consequently replacing CORBA is a long-term commitment.

- Complexity of the evaluation: the measure of how likely a wrong product will be selected [16]. This evaluation is far from trivial because of high amount of components affected by the selection and the complexity of the organizations involved (e.g., CTA), their processes and the system context. The stakeholders are distributed geographically across different continents and logically working in different subsystems so we can't fully determine their expectations. Moreover the evaluation team is under qualified in terms of experience and the large range of skills that are necessary for making this complex evaluation.

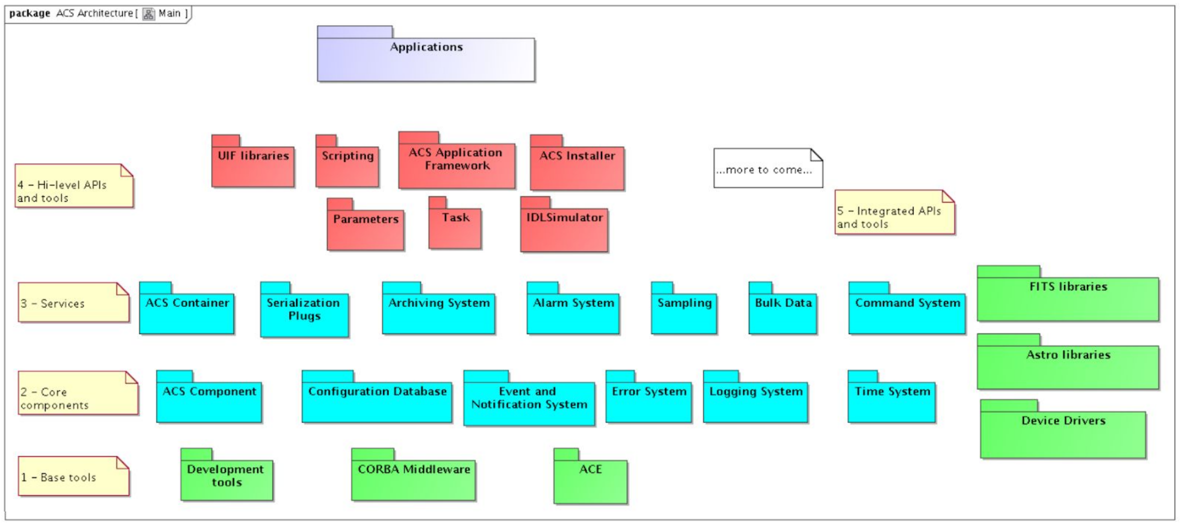


Figure 4.1: ACS packages.
Source: [8].

The selection of a communication middleware for the distributed control system of modern observatories such as CTA is a considerable and long-term investment. For these reasons a methodical selection process is required. Now that the depth of evaluation has been determined is time to pick the evaluation approach.

Best fit vs first fit

The *first fit* approach selects the first technology that complies the system needs [16]. On the other hand the *best fit* approach compare several alternatives to find the best. In the context of this evaluation should be executed because the following reasons:

- We get a considerable benefit if we acquire more than the minimum required quality and features [16]. The current ACS implementation might (arguably) be considered a first fit, but CORBA is struggling to fulfill the requirements of several organizations including ALMA and CTA. The best fit approach allow us to find a middleware that is able to fulfill the increasingly stricter requirements of modern observatories.
- It is cost effective. Investing some resources in the short term to choose the best middleware for CTA will reduce the risk of the system failing at latter stages of the software development. Furthermore, the impact of the communication middleware on the system architecture is so high that choosing the best one will potentially reduce the long-term development and maintenance cost of the ACTL system.

Filters

There are a lot of potential candidates for replacing CORBA. Many of them implement different communication paradigms. The evaluation cost increases with the number of candidates to be evaluated, therefore it is advisable to filter the initial list of candidates so we can discard early the clearly unsuitable alternatives [16]. The filters chosen for this evaluation are the following:

- Does the middleware provide an implementation of the request-reply communication pattern?
- Does the middleware supports events?
- Does the middleware support a scripting language (Python)?
- Is the middleware compatible with Linux Red Hat Enterprise recompiled distributions such as Scientific Linux and CentOS?
- Does the middleware have an open source implementation?
- Does the middleware provide the right level of abstraction?
- Is the middleware independent from text-based serialization formats?

More details about the filtering can be found in Section 4.3.1.

4.1.5 Estimating resources and schedule

As we determined in Section 4.1.3, a methodical evaluation with a best fit approach is required. The high risk and the complexity of the evaluation require it to be rigorous. The number of potential candidates is very high. A wide range of communication paradigms for distributed system and implementations are available in the market, so a lot of technologies should be evaluated. This also means that the evaluation has to be executed by a team of experienced evaluators with diverse skills (e.g., technical experts, domain experts, security professionals and end users) [16].

For CTA, the evaluation would have a high short-term cost because the organization should allocate several resources to allow a team of experts evaluate in detail a long list of technologies. Nevertheless if they can afford it then it will pay off because a rigorous evaluation reduces the long-term cost of the system which is very important in a big and long-term project such as CTA. Besides choosing the best middleware for their needs, the information generated from this process may be useful during system architecting, design and integration [16]. More importantly the evaluation reduces the risk of the system ultimately failing a later stages of the project when it is more expensive to make corrective actions.

In this document PECA with a best fit approach is executed. Although I do not have the resources to do a complete evaluation (e.g., collect data about all the relevant evaluation criteria) this is not a major problem because of the iterative nature of PECA. New iterations of the process should be executed over time when the system context changes (e.g., new technologies appears, change of the requirements, more resource allocated for evaluation, etc) [16]. For these reasons the present document will have an educational approach so it can serve as an input for future evaluation efforts as a first PECA iteration and as a guide for executing the process.

This iteration of PECA will consider a bounded number of candidates. The general strategy is described as follows:

- Select a candidate pool by doing a quick research about the different technologies used in the present for communication in distributed systems.
- Use the coarse filters (easy and cheap) introduced briefly in Section 4.1.3 to reduce the pool of potential candidates.
- Continue evaluating the remaining (i.e., most promising) candidates with more expensive criteria.

The resources destined to this evaluation are the following:

- 70 working days from the author.
- Physical resources for a hands-on experiment that are specified in Section 4.3.3.

These resources are clearly insufficient for a full execution of PECA to select the appropriate middleware product. Fortunately they are enough for showcasing the process and creating a guide, which is the objective of this iteration.

4.2 Establishing the criteria

This is the second step of PECA. As described in Chapter 3, I used the hierarchical decomposition method detailed in [35] for doing the first two tasks of this PECA step: defining the evaluation requirements and defining the evaluation criteria. Then I finish the step by prioritizing the criteria using pairwise comparisons detailed in [36]. An overview of the hierarchical decomposition method is presented in Section 2.6.1.

4.2.1 Analyse the influencing factors

In this section I present a brief analysis of the factors that determine the reuse goals of the middleware:

- Application requirements: these requirements were extracted from the requirement specification of ACTL [4]. The requirement specification of ACS for ALMA [44] was also analyzed to gain better understanding of the problem domain. ACS is a software framework created to support for the development of a distributed control and monitoring system. It has to provide communication capabilities and additional relevant services with the required level of quality (e.g., availability, reliability and performance) to accomplish the requirement specification. Choosing the appropriate communication middleware is key for ACS being able to fulfill increasingly demanding requirements.
- Application domain and architecture: ACS is the core framework of the distributed system in the ALMA observatory which implements a distributed object architecture promoted by the use of CORBA as its communication middleware. CTA will have high bandwidth networks (e.g., optic fibers of 10 GB/seg each), high throughput computing and CPU-intensive data processing and analysis. To tackle the heterogeneity of the system the software architects proposed the use OPC UA middleware for interoperability of the devices and the use of software bridge pattern for the communication with ACS components [15]. It is expected that ACS will take advantage of these conditions. Unfortunately it appears that CORBA does not fit the architecture very well, for example it provides unnecessary interoperability that is covered by the use of OPC UA. Also some studies report that CORBA is inefficient in high-speed networks [45].

  Regarding the application domain, ACTL is a soft real-time system because the correctness of its operations depend on the correctness of its outputs and their timeliness. Moreover CTA wants to maximize the use of the observatory operational time and the scientific return. These expectations are source of the high reliability and availability requirements of ACTL. More information about this factor can be found in the ACTL architecture document [15].
- Project objectives and constraints: ACS was built with the purpose of standardize and facilitate the software development in a heterogeneous environment with a development team scattered around the world with different development cultures. The ACTL team incorporated the use of software framework such as ACS and OPC UA as a mean to reduce development cost and the maintenance effort [15]. This is particularly important for CTA which wants to build ACTL to satisfy "the need to develop, maintain, and operate a more complex and more stable (when compared to existing IACT arrays) software system with limited manpower at moderate cost" [15, p. 5]. For CTA South the manpower cost represent approximately the 78% of the total cost of the project. In their risk registry the lack

of manpower, which is also too distributed and difficult to coordinate, is the cause of their top-priority risks. Moreover, CTA will go through a telescope deployment phase of several years so ACTL should be flexible and configurable enough to support an incremental deployment [15, p. 13].

- Availability of features: analysing this factor allow us to avoid potential unrealistic expectations to affect negatively the evaluation process. For instance, Does exist a middleware product that allows high level programming and provides common high quality services for supporting the control system, but that also provides a performant enough communication protocol implementation for the bulk data transfer needed by CTA (i.e., 77GB/seg approximately for all cameras)? This suspicion is based on the trade-off between performance and high level abstraction. The CTA architecture documentation proposes that the high throughput data transport might be based on a ZeroMQ library [15]. This technology would force de developers to work in a relative low level of abstraction. This factor should be continuously checked as we start collecting data from potentials candidates in the next step of PECA.
- Organization Infrastructure: Any organization using this approach should consider their own reuse infrastructure and reuse maturity. The organization interest, experience, commitment and skill in the software reuse are key in its success to evaluate and use OTS components [35]. I haven’t got access to this information regarding the ACTL team. Nevertheless, this document might help organizations, specially modern observatories, interested in creating or improving their own reuse infrastructure.

4.2.2 Identify and state the reuse goals

After analyzing the factors described above the evaluators can define the reuse goals. These goals represent the needs of the organization that should be satisfied by the candidates to be evaluated [35]. The reuse goal statement (high level description) for this iteration is the following:

The target application of the reuse software is the distributed monitoring and control system of CTA (ACTL). The OTS software being evaluated is a communication middleware technology that must be integrated into the ACS framework so it can support the development of the distributed system by providing standardized communication and high level services to the system components. The expected benefits (abstract and simple) are summarized in Table 4.3.

Possible constraints for OTS reuse: there are not many restrictions for the reuse of OTS software in this evaluation. Nevertheless, the ACTL software “prescribes the use of software frameworks, the application of widely accepted standards, tools and protocols, and follows basically an open-source approach” [15, p. 5].

Table 4.3: Categorized reuse goals.
Source: Elaborated by the author

Reuse Goals
Product Characteristic Goals
Communicate components in the distributed system.
Provide support for ACS common services such as monitoring, logging and alert services.
Enable remote control of the system (e.g., from the Internet).
Enable high reliability and availability of the system.
Provide secure communication between subsystems and components.
Enable high system performance.
Communicate an increasing amount of components (devices) in the system without having a significant negative impact in the performance (scalability).
Development Process Goals
Reduce the inherent complexity of the distributed system.
Provides a common platform/standard that contributes to unify the heterogeneous and distributed development environment.
Reduce the overall development effort required to build the system.
Support an iterative and incremental software development of the system.
Maintenance process goals
Reduce the risk of developing an unmaintainable complex system.
Reduce the overall maintenance cost.

Cost budget for the use of OTS software: Even though this information is not relevant to this execution of the process, in a next iteration, the ACTL team should state how much they are able to spend to replace CORBA with a new middleware. Because the OTS software is open source, the cost of using it can be translated into the selection and integration effort required which might be measured in several units such as staff-hours [46]. Considering the project objectives and constraints discussed above we might expect that CTA would consider replacing CORBA if the following conditions are met:

- The cost is affordable in the short-term. Probably some ACS services (based on CORBA services) will have to be implemented again using the available features of the CORBA replacement.

- The redesign of ACS and the integration of the new middleware do not hinder the compliance of the ACTL project deadlines.
- The benefit in the long term (e.g., increased system quality, reduced development effort and maintenance effort), overshadows the short term cost.

4.2.3 Identifying and formulate evaluation goals

As introduced in Section 2.6.1, the evaluation goals represent the objectives of the evaluation process. We derive them from the reuse goals and formulate them using the Goal-Question-Metric syntax [35]. In Table 4.4 one of the many evaluations goals for this iteration is presented.

Table 4.4: Performance as evaluation goal with GQM notation.
Source: Adapted from [35].

ID	QC-02
Object (Entity)	Communication middleware
Focus (Issue)	Performance
Purpose	Evaluate
Point of View	-

As pointed out in [35] the object is the entity being analyzed and the focus is the attribute of interest. The object and focus can be extracted directly from the respective reuse goal in Table 4.3. The purpose is almost always evaluate, but in some cases it might be simple characterization to understanding or even prediction. The point of view field is relevant when two different stakeholders are interested in the evaluation goal and their views have to be considered. I added the ID field to keep track of the evaluation goal during the decomposition.

We can see another example in Table 4.5. If we go back to the influencing factors we can find that the project highest risks are originated by limited manpower. This lack of resources was the source of several reuse goals included in Table 4.3 under the *development process* and *maintenance process* categories. In similar fashion these reuse goals generated the need to evaluate the impact of the communication middleware on the manpower efficiency. As the reader might notice, this evaluation goal is still too abstract to be easily measurable.

Table 4.5: Manpower efficiency as evaluation goal with GQM notation.
Source: Adapted from [35].

ID	SC-02
Object (Entity)	Communication Middleware
Focus (Issue)	Manpower efficiency support
Purpose	Evaluate
Point of View	-

4.2.4 Define high level criteria

The next task of the hierarchical decomposition method is to decompose each evaluation goal in "a set of high level criteria that characterize it" [35, Sec. 3.2]. Continuing the examples used in Section 4.2.3, I decomposed the evaluation goals into high level criteria. The performance of the middleware can be characterized by the following criteria:

- Time behaviour: "the capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions" [47, Sec. 6.4.1].
- Resource utilization: "the capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions" [47, Sec. 6.4.2].

The impact of the middleware in manpower efficiency might be characterized by the following criteria:

- System extensibility support: the middleware contribution to the ability to extend the system and the effort required to do it. Extensibility effectively reduces the development and maintenance effort. It also increases the reusability of the components of the system.
- System portability support: the middleware contribution to the ability to port components and services to different platforms and the effort required to do it.
- Reuse of legacy components: the middleware enables the reuse of legacy component such as the GUI developed by ALMA.
- System transparency support: the middleware hides some of the complexity of the distributed system reducing the cognitive burden of the developers and thus the development effort.

4.2.5 Can we easily measure the criteria?

The reader might notice that the criteria presented in Section 4.2.4 are still too abstract to be measured in a straightforward manner. For that reason, as suggested in Section 2.6.1, we must continue decomposing it. For example let's focus on *time behaviour* and *system extensibility support*. The first one is easy to decompose, as we typically see in middleware benchmarking, in two criteria:

- Latency: "the amount of time it takes for a client to invoke a two-way operation in a server and receive the results of the operation" [48, p. 3]. We are particularly interested on the overhead introduced by the middleware.
- Throughput: "how much data can move the middleware per unit of time" [48, p. 6].

The second one is more tricky. The extensibility of a distributed system is improved by high cohesion and low coupling. Cohesion can be defined as "the manner and degree to which the task performed by a single software module are related to one another" [49, p. 17]. One can argue that cohesion is application dependent and it is not directly related to the middleware used to enable communication between components. Nonetheless the coupling of a distributed system is affected by the choice of communication middleware as described in [50]:

- Default coupling: the baseline coupling introduced to the system by the communication middleware. This coupling depends on the interface types provided by the middleware.

We can measure the subcriteria generated in this section directly so we register them as evaluation attributes. Latency and throughput can be measured quantitatively while default coupling can be determined qualitatively through a description of the interfaces of the middleware as detailed in Section 4.3.2.

Then the evaluators have to continue decomposing the remaining high level evaluation goals and criteria. We can see the full decomposition of the evaluation goal QC-02 in Table 4.6.

Figure 4.2 shows the result of the hierarchical decomposition method. The red numbers are the priorities of the criteria that were generated after the decomposition (see Section 4.2.6). This hierarchy represent the vision that the evaluation team has about the system and its needs regarding the selection of the communication middleware. It is important to note that two different evaluation teams may arrive to different representations of the same problem due to potential differences in their knowledge and experiences. In this PECA execution I attempted to compensate my lack of experience with an extensive research about distributed systems, open source software and the ACTL system.

Section 5.2 provides some guidelines and recommendations to assist the evaluators in choosing good evaluation criteria.

Table 4.6: Full decomposition of an evaluation goal.
Source: Elaborated by the author

Evaluation Goal	
ID	**QC-02**
Object (Entity)	Communication Middleware
Focus (Issue)	Performance
Purpose	Evaluate
Point of View	
Evaluation Criteria	
QC-02-01	Time behaviour
Definition	The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.
QC-02-01-01	Latency
Definition	The amount of time it takes for a client to invoke a twoway operation in a server and receive the results of the operation.
Measurement	Number of interaction per second
QC-02-01-02	Throughput
Definition	How much data can move the middleware per unit of time
Measurement	Mbits/s of data moved by the middleware
QC-02-02	Resource utilization
Definition	The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.
QC-02-02-01	CPU consumption
Definition	Number of CPU cycles spent by the middleware in order to marshall and unmarshall the data. Complex serialization methods consume more CPU.
Measurement	% of time spent in the middleware for a particular workload.
QC-02-02-02	Network bandwidth usage
Definition	The amount of bytes that the middleware add to the message's payload increase the bandwidth usage. Simple serialization methods tend to use more bandwidth.
Measurement	Serialization overhead (MB)
QC-02-02-03	Memory consumption
Definition	The amount of memory needed by the middleware. Complex encoding tends to need more supporting data structures than simple encodings. A non-optimal buffer size can reduce performance.
Measurement	Working set size (MB)

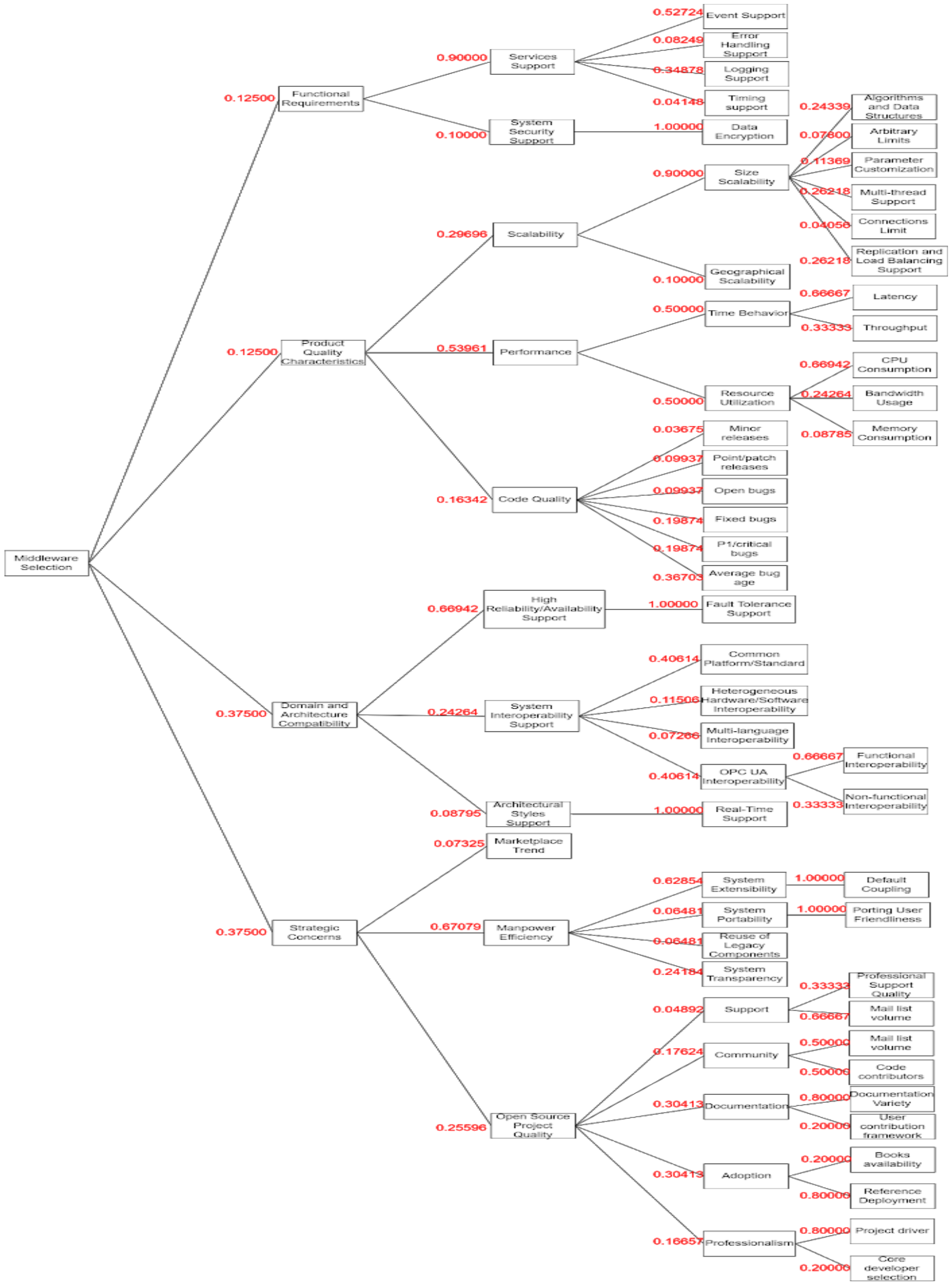

Middleware Selection
0.12500 Functional Requirements
0.90000 Services Support
0.52724 Event Support
0.08249 Error Handling Support
0.34878 Logging Support
0.04148 Timing support
0.10000 System Security Support
1.00000 Data Encryption
0.12500 Product Quality Characteristics
0.29696 Scalability
0.90000 Size Scalability
0.24339 Algorithms and Data Structures
0.07800 Arbitrary Limits
0.11369 Parameter Customization
0.26218 Multi-thread Support
0.04056 Connections Limit
0.26218 Replication and Load Balancing Support
0.10000 Geographical Scalability
0.53961 Performance
0.50000 Time Behavior
0.66667 Latency
0.33333 Throughput
0.50000 Resource Utilization
0.66942 CPU Consumption
0.24264 Bandwidth Usage
0.08785 Memory Consumption
0.16342 Code Quality
0.03675 Minor releases
0.09937 Point/patch releases
0.09937 Open bugs
0.19874 Fixed bugs
0.19874 P1/critical bugs
0.36703 Average bug age
0.37500 Domain and Architecture Compatibility
0.66942 High Reliability/Availability Support
1.00000 Fault Tolerance Support
0.24264 System Interoperability Support
0.40614 Common Platform/Standard
0.11506 Heterogeneous Hardware/Software Interoperability
0.07266 Multi-language Interoperability
0.40614 OPC UA Interoperability
0.66667 Functional Interoperability
0.33333 Non-functional Interoperability
0.08795 Architectural Styles Support
1.00000 Real-Time Support
0.37500 Strategic Concerns
0.07325 Marketplace Trend
0.67079 Manpower Efficiency
0.62854 System Extensibility
1.00000 Default Coupling
0.06481 System Portability
1.00000 Porting User Friendliness
0.06481 Reuse of Legacy Components
0.24184 System Transparency
0.25596 Open Source Project Quality
0.04892 Support
0.33333 Professional Support Quality
0.66667 Mail list volume
0.17624 Community
0.50000 Mail list volume
0.50000 Code contributors
0.30413 Documentation
0.80000 Documentation Variety
0.20000 User contribution framework
0.30413 Adoption
0.20000 Books availability
0.80000 Reference Deployment
0.16657 Professionalism
0.80000 Project driver
0.20000 Core developer selection

Figure 4.2: Hierarchical representation of the MCDM problem and the local priorities of each criterion.
Source: Elaborated by the author

4.2.6 Establishing priorities

After generating a hierarchical structure to represent the decision-making problem, the evaluators should assign priorities to the criteria to reflect the fact that not all the criteria are equally important. The hierarchical decomposition method and PECA propose the use of Analytic Hierarchy Process (AHP) to accomplish this task. The rationale of the use of AHP in the context of this evaluation is presented in Chapter 3.

For this task we just need the pairwise comparison technique described in AHP. Once a hierarchy was generated the next step in AHP is to compare the criteria in pairs using judgments that are "based on knowledge and experience to interpret data according to their contribution to the parent node in the level immediately above" of the hierarchical structure [36, Sec. 3]. The judgments are made using the fundamental scale presented in Table 4.7. This scale enable the evaluators to map a verbal statement to a natural number between 1 and 9. In pairwise comparisons we assign the value 1 to the less important criterion and then we make a verbal statement about the level of dominance of the most important criterion in the pair [36].

Table 4.7: Saaty Fundamental scale.
Source: Adapted from [51]

The Fundamental Scale for Pairwise Comparisons		
Intensity of importance	**Definition**	**Explanation**
1	Equal importance	Two elements contribute equally to the objective
2	Equal to moderate importance	
3	Moderate importance	Experience and judgment slightly favor one element over another
4	Moderate to strong importance	
5	Strong importance	Experience and judgment strongly favor one element over another
6	Strong to very strong importance	
7	Very strong importance	An element is favored very strongly over another, its dominance demonstrated in practice
8	Very strong to extreme importance	
9	Extreme importance	The evidence favoring one element over another is of the highest possible order of affirmation
Reciprocals of above	If element *i* has one of the above nonzero numbers assigned to it when compared with element *j*, then *j* has the reciprocal value when compared with *i*	A reasonable assumption
Rationals	Ratios arising from the scale	If consistency were to be forced by obtaining *n* numerical values to span the matrix

The judgements should be registered because they eventually must be processed mathematically to generate the priorities of the criteria. This can be done by using a spreadsheet or a specialized software for AHP.

The reader should understand that these priorities depend on the evaluation team and the context in which the judgements are made because prioritization of the criteria is an inherently subjective task. Changes on the evaluation team or the system context could potentially generate variations on the judgements if this process is repeated in the future.

I followed a top-down approach by making pairwise comparisons about the importance of the four types of criteria proposed by the hierarchical decomposition method (i.e., functional requirements (FR), product quality characteristics (QC), domain and architecture compatibility (DA) and strategic concerns (SC)) regarding the goal which is the selection of a suitable communication middleware for the ACS framework and CTA (see Figure 4.2). A summary of these pairwise comparisons (PC) is presented in Table 4.8.

Table 4.8: Pairwise comparisons regarding the goal using the Saaty scale.
Source: Elaborated by the author.

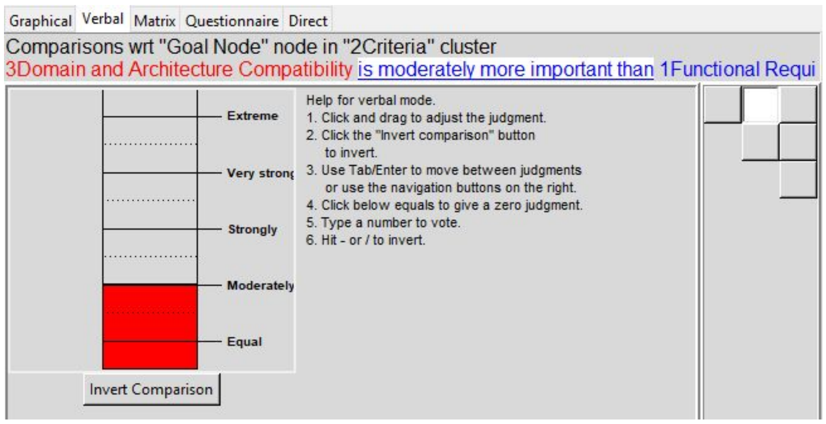
PC #1	FR	1	QC	1
Functional requirements supported by the middleware are important because they have a considerable impact not only in the development effort but also in the performance and reliability of the services. One important reason for using the ACS framework are its standardized services. Without communication middleware services the ACTL developers would have to reimplement the ACS services from the scratch struggling to achieve their required levels of performance and reliability.				
Product quality characteristics are important because they reflect the maturity of the implementation of the services and communication protocols which is an internal metric of its reliability. If the middleware services are not reliable enough they are useless. Moreover we would want to improve the current performance of ACS while maintaining its scalability. Nevertheless ACS has already being used successfully in end-to-end prototypes.				
Functional Requirements are equally as important as *Product Quality Characteristics.*				
PC #2	FR	1	DA	3
Functional requirements importance is described as above.				
Domain and architecture compliance is important because it addresses one of the hardest requirement of the system: high reliability and availability to maximize the scientific return. Not only the middleware should be reliable, it also should provide additional features such as fault tolerance support for achieving these domain related goals. CTA has put considerable effort in the design of the ACTL architecture to tackle the great heterogeneity of the system. Even though the software and hardware heterogeneity is covered by OPC UA, programing language interoperability is still a desirable feature. More importantly the middleware should serve as a platform to unify heterogeneous cultures of geographically dispersed software development teams, partially mitigating one of the top priority risk of the ACTL software system [15].				
Domain and Architecture Compliance is moderately more important than *Functional Requirements*.				

Table 4.8: Continued.

PC #3	FR	1	SC	3
Functional requirements importance is described as above. Strategic concerns are important because they address many high priority risks of the ACTL software related to the limited manpower available for developing a complex system with high quality requirements in a timely fashion. They also address the fact that the ACTL system will be operated and maintained by a reduced workforce. Finally strategic concerns also takes the marketplace trend and the open source project supporting the middleware into consideration which are important reasons to replace CORBA. *Strategic Concerns* are moderately more important than *Functional Requirements.*				
PC #4	QC	1	DA	3
Good product quality characteristics are desirable and they indirectly affect availability of the whole system. Nevertheless the domain and architecture compatibility addresses the availability of the system directly and also the architecture based hard constraints such as the use of OPC UA middleware. *Domain and Architecture Compatibility* is moderately more important than *Product Quality Characteristics*.				
PC #5	QC	1	SC	3
The importance of the product quality characteristics is described as above. Meanwhile the Strategic Concerns address the feasibility of developing and maintaining the system which are high priority risks. *Strategic Concerns* are moderately more important than *Product Quality Characteristics*.				
PC #6	DA	1	SC	1
Both criteria deal with high priority risks of ACTL, the feasibility and success of the whole project. *Strategic Concerns* address several high priority risks while *Domain and Architecture Compliance* also deals with hard constraints. *Domain and Architecture Compliance* is equally as important as *Strategic Concerns.*				

A summary of this type is not an essential part of AHP, but it may be useful to record the rationale of the judgments. Pairwise comparisons help to deal with the inherent subjectivity of assigning priorities by making the judgments more defensible and accurate. Furthermore they also show the insight, experience and knowledge of the evaluators [52].

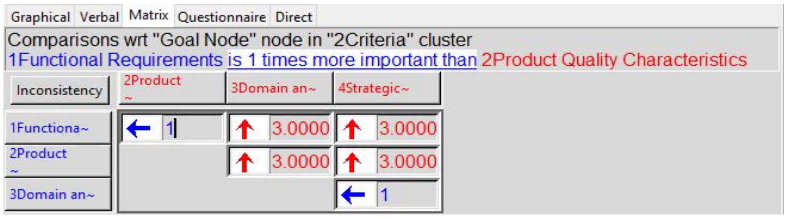
After we do all the required pairwise comparisons we can use a spreadsheet or an AHP decision making software to derive the priorities. In this iteration I entered the judgements in the software Super Decisions v3 for automatic calculation of the priorities of the criteria. Super Decisions v3 is the only free educational software that implements AHP. It was developed by the team of the the creator of the process. It provides various forms for entering the judgments. Figure 4.3 and Figure 4.4 show two of the five modes for entering the judgements in Super Decisions v3.



Comparisons wrt "Goal Node" node in "2Criteria" cluster
3Domain and Architecture Compatibility is moderately more important than 1Functional Requi

Help for verbal mode.
1. Click and drag to adjust the judgment.
2. Click the "Invert comparison" button to invert.
3. Use Tab/Enter to move between judgments or use the navigation buttons on the right.
4. Click below equals to give a zero judgment.
5. Type a number to vote.
6. Hit - or / to invert.

Figure 4.3: Verbal mode for entering judgements in Super Decisions v3.
Source: Elaborated by the author

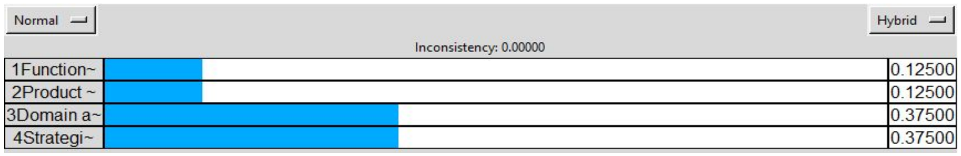


Comparisons wrt "Goal Node" node in "2Criteria" cluster
1Functional Requirements is 1 times more important than 2Product Quality Characteristics

	2Product ~	3Domain an~	4Strategic~
1Functiona~	← 1	↑ 3.0000	↑ 3.0000
2Product ~		↑ 3.0000	↑ 3.0000
3Domain an~			← 1

Figure 4.4: Matrix mode for entering judgements in Super Decisions v3.
Source: Elaborated by the author

The resulting local priorities are automatically obtained as is shown in Figure 4.5. The software also calculates an inconsistency index. This index measures the transitivity and consistency of the judgements. In this example the judgments are perfectly transitive so the inconsistency index is zero. Saaty [36] state that the value of this index has to be less than 0.1 for the results being meaningful. If the inconsistency of the judgments is too high, the evaluators have to review them and after a discussion they have to carefully modify them.



Inconsistency: 0.00000

1Function~	0.12500
2Product ~	0.12500
3Domain a~	0.37500
4Strategi~	0.37500

Figure 4.5: Local priorities of the criteria.
Source: Elaborated by the author.

The local priorities of each node regarding to its parent node are shown with red numbers in Figure 4.2. The evaluators will need these priorities during the consolidation of data in the *Analysing the data* step of PECA. For now the reader can notice what are the most important criteria at each level of the hierarchy according to me, based on the analysis of the the ACTL software system documentation and its context.

In this section the prioritization was made by only one evaluator. Section 5.3 shows some recommendations for conflict resolution and group decision-making.

4.3 Collecting the data

One might argue that the main objective of this step is to gather the necessary data that allows the evaluators to discriminate how each candidate perform in respect to the previously defined evaluation criteria. This is not necessarily true, because collecting data also increases the understanding of the evaluators about the product marketplace and system context [16]. If we recall the previous steps of PECA there are some task that are dependent on the evaluators experience and knowledge. During the *Collecting the data* step the evaluators knowledge improves. They realize how far their understanding was from reality and they might discover some unpleasant facts [16]:

- Product capabilities different from the evaluation team expectations: for example, I expected that ZeroMQ was going to outspeed the other candidates in almost every scenario. This expectation was proven wrong in my latency benchmarking described in Section 4.3.3.
- Unexpected interactions and architectural mismatches: An interesting concern might be the interoperability between the candidates and OPC UA middleware. An outstanding communication middleware that has problems to work with OPC UA (functional and non-functional interoperability) will generate an architectural mismatch that must be addressed.

These unpleasant findings are good because PECA is an iterative process. With this improved knowledge and more accurate understanding we can start a new iteration of PECA to get more relevant and precise results.

Section 5.4 present some tips to choose adequate techniques for collecting data.

4.3.1 Quick Assessment (Filtering)

Even though we might consider this task fairly straightforward, in this evaluation I followed the guidelines proposed by Wasserman, Pal and Chan [39] for executing an effective quick assessment which are described in Section 2.6.3.

In this work I am interested on evaluating the communication middleware alternatives for the ACS framework so it can be used for the development of ACTL system. Therefore, in this case, it is clear that the target usage of the product is for development purposes. For simplicity and for maximum filtering of the candidates I chose a policy of *all positive* for passing the filtering phase (i.e., only the middleware candidates that satisfy all the viability indicators will be further evaluated) .

In [39] a set of reusable viability indicators are provided. Nevertheless the evaluators are encouraged to use indicators that are relevant in their system context. It is fundamental that these indicators are easily measurable so the quick assessment can be indeed quick. One example of an important viability indicator that is not easily measurable by me is the license compatibility of the middleware with the ACTL project. Below I present the viability indicators chosen for this quick assessment, describe their importance and the policy for passing each one.

Does the middleware provide an implementation of the request-reply communication pattern?

ACTL has to provide a framework to control all telescopes and auxiliary systems on a CTA site. Specifically two basic interactions are covered by this viability indicator:

- Command delivery: “The control system should be able to deliver commands to the process in charge of the target device and the command delivery should be reliable i.e., commands must always be acknowledged” [4, p. 10].
- Query for data: a client application can request data from the process in charge of an specific device. For example, an operator may ask for the temperature value of a thermometer using a GUI.

In this assessment we are looking to filter potential candidates that could eventually replace CORBA as the main communication middleware of ACS. The basic functionality expected by such a middleware is the implementation of the required patterns and protocols that allow to communicate remote processes.

There are many ways to share data, but there are only two main integration styles for sharing functionality: remote procedure call and messaging [53]. There are some trade-offs between these two styles, but they both support the interactions mentioned above by implementing the request-reply pattern [53, 54]. A message based middleware will probably need two point-to-point channels: one for sending the request (i.e., a command or query) and other for receiving the reply (i.e., a return value or result of the query). Any candidate that provides a request-reply protocol implementation over TCP will pass this filter.

Does the middleware supports events?

In addition to control, ACTL has to provide a framework to monitor the functioning of all instruments and handle errors, alerts and warnings from instruments in the CTA sites. Events are a good way for transmitting alerts, warnings and exceptions. Events are commonly supported by an implementation of the publish-subscribe pattern [53, 55] which is also very useful for sampling the data of the different devices for monitoring purposes.

If the main communication middleware does not support events then other technologies will have to be integrated into ACS to cover this requirement (e.g., a secondary middleware such as DDS in the ALMA implementation of ACS). These technologies “can be expensive, can lead to vendor lock-in and can increase the learning curve for developers” [53, p. 40] effectively increasing the complexity of the system and the burden on developers.

Garcia [54, p. 33] defines event-based middleware as “a middleware for large-scale distributed systems that implements publish-subscribe communication between components, providing a scalable and efficient event service.” For the purpose of this assessment any candidate that provides a publish-subscribe protocol implementation or a more sophisticated event service will pass this filter.

Does the middleware support a scripting language (Python)?

The middleware should support a scripting language that can serve as a base for the ACTL scripting language which will be used to “define observing scripts, as well as to serve as a suite of interactive commands to be used by hardware engineers for testing or debugging the equipment, or by operators for developing new observation procedures” [15, p. 30]. This indicator is important because ACTL should support “scripting of sequences of control actions” as stated in the requirement specification [4, p. 11].

Nonetheless CTA haven’t specified which features are looking in the scripting language. The ACTL team is considering the use of the ALMA Control Command Language (CCL) which is a simple Python wrapper of the Control Software. Python was chosen as the base of the CCL because it has the following characteristics [8, Sec. 3.2.1]:

- Object oriented
- Platform independent
- Provides flow control, variables and procedures
- Plug and play to install scripting language
- Connects to CORBA
- Easy to use for non-programmers
- Rapid prototyping similar to programming languages used in the project
- GUI development
- Embeddable into code for interactive applications

Any middleware that supports Python or another similar scripting language will pass this filter.

Is the middleware compatible with Linux Red Hat Enterprise recompiled distributions such as Scientific Linux and CentOS?

The ACTL team has already made several architectural decisions about the ACTL system. The operating system for all servers will be based on Red Hat Enterprise Linux (RHEL) recompiled distributions (e.g., Scientific Linux and CentOS) [15]. Wegner et al. [56, p. 4] reported that the ALMA ACS distributions are executed on a RHEL re-distribution which will therefore be used on all full-scale ACTL computers.

RHEL has been developed for long-term supported commercial usage. Scientific Linux (SL) as a RHEL re-compilation has successfully been used inside Particle and Astroparticle Physics projects for many years such as the Large Hadron Collider (LHC). Furthermore, "Inside CTA, important parts of the on-line and off-line software is already running under SL, e.g., the Grid middleware and the Alma Common Software" [15, p. 47].

Any middleware that supports Red Hat Enterprise Linux will pass this filter.

Does the middleware have an open source implementation?

The ACTL team didn't plan any software licence cost yet because the approach taken by ACTL is to use open source software, whenever possible [15].

The usage of open source software may provide several advantages to CTA. First of all it helps to avoid vendor lock-in. Moreover by having a community continuously fixing and improving the middleware, the use of open source software will reduce the burden of ACTL developers and maintainers which can focus on improving the ACTL specific software. Third party software lifespan is difficult to evaluate [15] and the use of open source software allows the ACTL team to take over the maintenance in case of need.

Any middleware supported by an active open source project will pass this filter.

Does the middleware provide the right level of abstraction?

Section 2.3 provides a taxonomy of middleware paradigms elaborated by Alrahmawy [25]. This classification is given in order starting from the lowest level of abstraction to the highest.

"The concept for the ACTL software accounts for the need to develop, maintain, and operate a more complex and more stable (when compared to existing IACT arrays) software system with limited manpower at moderate cost" [15, Sec. 1.1]. To achieve this the developers should be able to work at a high level of abstraction so the development and maintenance effort required can be effectively reduced. On the other hand ACTL software has high performance requirements that demand to work at a low level of abstraction, but low-level technologies reduce the system reliability, flexibility and reuse [45]. A balance is required in the abstraction and optimization trade-off.

ACS was built on top of CORBA which implements the Object Request Broker paradigm. Moreover ALMA developed its own custom CCM using CORBA increasing the level of abstraction even further [7]. In doing so the ACS framework enforces the Component Oriented Architecture Paradigm and it is struggling to meet the performance requirements. It might be a good policy to stop looking for middleware with a higher level of abstraction than ACS. On the other hand some projects working in the same area as ACS, such as CERN's Controls System [10, 57] and TANGO, decided to replace CORBA with ZeroMQ which provides a lower level of abstraction. While CERN expects to replace CORBA completely TANGO [58] uses CORBA for point-to-point communication and ZeroMQ for publish-subscribe communication. Therefore we can use the paradigm implemented by ZeroMQ as a conservative lower bound for the right level of abstraction.

Any middleware that is between the fourth and seventh level of abstraction will pass this filter (see Section 2.3).

Is the middleware independent from text-based serialization formats?

The middleware should implement its own serialization of messages or at least it should be compatible with a third party serialization library. In any case it must not rely on text-based serialization formats like XML as its only serialization format.

Even though XML enables easy and powerful integration between systems, its drawbacks make it unsuitable for ACTL. XML as a tag-based language is usually “bloated with long named tags, complex data structures and big amounts of plain texts data” making its serialization and deserialization “complex, heavy and slow” [59, p. 2]. Because XML is a text based format, XML messages are usually very large in size compared to the original data requiring more bandwidth to transport them.

The middleware will pass this filter if it doesn’t rely exclusively on text-based serialization for transport.

Another indicator that didn't make it into the quick assessment

It can be argued that support for bulk data transfer is also a communication concern and should be used a an initial filter in the quick assessment. I didn’t include bulk data transfer support for two reasons:

- Hard to evaluate: it would require a throughput benchmarking with a simulation of the real system. At least real camera data should be used.
- Unrealistic expectation: during the execution of PECA I came to the conclusion that it might be unrealistic to ask for a high level of abstraction middleware suitable for supporting the control system that also provides the very high performance required (throughput) for bulk data transfer in CTA. Using this filter could potentially eliminate too many candidates. Furthermore, CTA already proved the feasibility of using ZeroMQ or TCP for bulk data transfer [15] so an integration of another technology with ACS for accomplish this task is considered a viable solution.

Nevertheless this is an important criterion that should be evaluated for the candidates that pass the filtering phase.

Quick assessment results

Table 4.9 summarizes the result of the initial filtering (quick assessment) in which each candidate, of a selected middleware pool, was evaluated using the previously described filters (viability indicators):

A. Does the middleware provide an implementation of the request-reply communication pattern?
B. Does the middleware supports events?
C. Does the middleware support a scripting language (Python)?
D. Is the middleware compatible with Linux Red Hat Enterprise recompiled distributions such as Scientific Linux and CentOS?
E. Does the middleware have an open source implementation?
F. Does the middleware provide the right level of abstraction?
G. Is the middleware independent from text-based serialization formats?

Table 4.9: Quick assessment summary.
Source: Elaborated by the author

Technology	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**Score**
Open MPI [60]	✕	✕	✓	✓	✓	✕	✓	**4**
RTI Connext DDS [61]	✓	✓	✓	✓	✕	✓	✓	**6**
OpenDDS [62]	✕	✓	✕	✓	✓	✓	✓	**5**
ZeroMQ [63]	✓	✓	✓	✓	✓	✓	✓	**7**
YAMI4 [64]	✓	✓	✓	✓	✕	✓	✓	**6**
IBM MQ [65]	✓	✓	✓	✓	✕	✓	✓	**6**
Apache Thrift [66]	✓	✕	✓	✓	✓	✓	✓	**6**
gRPC [67]	✓	✕	✓	✓	✓	✓	✓	**6**
EPICS [68]	✓	✓	✓	✓	✓	✓	✓	**7**
Java RMI [69]	✓	✕	✕	✓	✕	✓	✓	**4**
ZeroC Ice [70]	✓	✓	✓	✓	✓	✓	✓	**7**
CORBA [26-28]	✓	✓	✓	✓	✓	✓	✓	**7**
Web Services[1]	✓	✓	✓	✓	✓	✕	✕	**5**
WCF [71]	✓	✕	✕	✕	✕	✕	✓	**2**
Apache CXF [72]	✓	✕	✕	✓	✓	✕	✓	**4**
Protocol Buffers[2] [73]	**N/A**	**N/A**	✓	✓	✓	**N/A**	✓	**N/A**

According to my previously chosen policy, from the 16 candidates only four passed the filtering phase: ZeroMQ, EPICS, Ice and CORBA (as is implemented in ACS).

4.3.2 Literature review about default coupling

The ISO/IEC/IEEE 24765 standard defines extensibility as “the ease with which a system or component can be modified to increase its storage or functional capacity” [74, p. 136]. Even though there is a trade-off between extensibility and performance, software developers build extensible software “mainly for reducing the cost of implementing new or similar functionality in a system” [75, Sec. 1.1.2]. This characteristic becomes especially

[1] Due to the great offer of technologies for implementing Web Services, in this quick assessment I take into account the things that most implementations do from a theoretical standpoint.

[2] Protocol Buffers is just for serializing structured data and thus it can not be compared directly with communication middleware alternatives.

beneficial when we develop large, complex and expensive distributed systems such as ACTL.

One critical factor for attaining high system extensibility is to promote low coupling. Yourdon and Constantine [76] introduced the concept of coupling as the amount of knowledge we need about one module to be able to understand another module. Wijegunaratne and Fernandez [50] adapted this notion of coupling to distributed systems. Moreover they recognize that there are two main sources of coupling in distributed systems: the middleware which provide the default coupling and poor design choices.

In a distribution environment the coupling of the system has an impact in the development and the maintenance. Some of these consequences are described in [50]. For example after designing a distributed system if we modify the external interfaces of a shared software component it may cause a ripple effect of changes across the entire system. Furthermore if the development team is geographically dispersed, these changes will also require that the potential conflicting parties renegotiate and reach consensus. Finally a component struggling to meet the required QoS may negatively affect other tightly coupled components and the development teams might be unable to solve the problem if the problematic component is out of their jurisdiction.

As the reader might already notice, knowledge is difficult to quantify. Fortunately for us Wijegunaratne and Fernandez [50] created a framework to assist evaluators in making effective decisions, such as choosing the correct middleware for the system, by describing general desirable properties that decrease the coupling in distributed systems. Specifically the middleware contributes to the coupling of the system in two major ways: “by being the repository of all or part of the administrative information, and secondly by extending one or more specific programming interface types, each with a default level of coupling to the application” [50, Sec 5.4]. The framework allowed me to define this evaluation attribute during the execution of the hierarchical decomposition method as described in Table 4.10.

Table 4.10: Definition of default coupling as an evaluation attribute.
Source: Elaborated by the author

SC-02-01-01	Default coupling
Definition	The baseline coupling introduced to the system by the communication middleware.
Description	Description of interfaces types and the administrative information management provided by the middleware.

This means that the data to be collected is a qualitative description. After studying the framework, I identified the data items that should be included in the description for each interface provided by the middleware [50]:

- Ease of the passage of control: communication interfaces that force or facilitate the passage of control from one system component to another are able to induce higher coupling than those who don't.
- Administrative information location: storing information such as the names of the components, their location (i.e., network address and protocol) and the users access authorization in the middleware induces less coupling than keeping this information at the application level.
- Administrative information fragmentation: storing the administrative information in a centralized location induces less coupling than fragmenting it in the distributed system.
- Communication types provided: the middleware provide interfaces which can be characterized as *available* or *non-available*, *conversational* or *non-conversational*, *synchronous* or *asynchronous*, *transactional* or *non-transactional* and *static* or *dynamic*. The coupling provided by a particular interface depends on the communication types associated with it.
- Binding: It is "the resolution of aspect of contacts between modules." In other words "is the act of associating some aspect of one software module with another remote module" [50, pp. 103-104]. The earlier the binding, the stronger the coupling between the application components. There are three types of bindings: Binding to form/structure, to implementation and to occurrence.

For gathering this data about the middleware I used several sources such as the official websites of the products, their user manuals and their API documentation. Then I summarized the collected data about the candidates and their communication interfaces in tables such as Table 4.11.

Table 4.11: Data collected about the ZeroC Ice default coupling.
Source: Elaborated by the author

ZeroC Ice	
Interface Type	**Static Invocation**
Easy passage of control	Average
Administrative information location	Ice stores the administrative information in the infrastructure. It provides two locations services: IceGrid and IceDiscovery.
Administrative information fragmentation	The administrative information is centralized in a location service.
Communication types	● Available ● Synchronous and Asynchronous ● Non-Conversational ● Non-Transactional ● Static
Binding to form	This binding occur early, at development time, when the software developers use Specification Language for Ice (SLICE) to define the interfaces that the applications are going to use. Then the SLICE compilers generate methods for synchronous and asynchronous invocations.

Table 4.11: Continued.

Binding to implementation	Thanks to SLICE the binding to implementation never happens because the developers can change the implementation language of any component to some other language supported by Ice using the SLICE compiler without having to modify the components with which interacts.
Binding to Occurrence	The binding to occurrence happens late because the components can ask the infrastructure (i.e., Ice location service), at runtime, for the administrative information required to connect with other components.
Interface Type	**Dynamic Invocation**
Easy passage of control	Difficult
Administrative information location	Ice stores the administrative information in the infrastructure. It provides two locations services: IceGrid and IceDiscovery.
Administrative information fragmentation	The administrative information is centralized in a location service.
Communication types	● Available ● Synchronous and Asynchronous ● Non-Conversational ● Non-Transactional ● Static (early binding to form)
Binding to form	This binding occur early, at development time. Even though the dynamic invocation interface allows to build the request at runtime, Ice doesn't provide a mechanism for discovering the remote interfaces during runtime so the components have to agree about the invocation format (name and list of parameters) at development time using SLICE.
Binding to implementation	The client and servers don't bind to implementation if we only use the languages supported by Dynamic Invocation (i.e., C++, Java and C#).
Binding to Occurrence	The binding to occurrence happens late because the components can ask the infrastructure (i.e., Ice location service), at runtime, for the administrative information required to connect with other components.
Interface Type	**IceStorm**
Easy passage of control	Difficult
Administrative information location	Ice stores the administrative information in the infrastructure. IceStorm can also persist administrative information about the publish-subscribe communication.
Administrative information fragmentation	The administrative information is centralized in an Ice location service. IceStorm maintains information about topics, links, and subscribers in a database.
Communication types	● Available ● Synchronous and Asynchronous ● Non-conversational ● Non-transactional ● Static (early binding to form)

Table 4.11: Continued.

Binding to form	This binding occur early, at development time. Developers have to define topic interface using SLICE which then are implemented by the subscribers as described in the Ice manual: A topic is essentially equivalent to an application-defined Slice interface: the operations of the interface define the types of messages supported by the topic. A publisher uses a proxy for the topic interface to send its messages, and a subscriber implements the topic interface (or an interface derived from the topic interface) in order to receive the messages. This is no different than if the publisher and subscriber were communicating directly in the traditional client-server style; the interface represents the contract between the client (the publisher) and the server (the subscriber), except IceStorm transparently forwards each message to multiple recipients [77, Sec. 44.3.2].
Binding to implementation	Similarly to the static interface, with IceStorm the publishers and subscribers never bind to implementation thanks to SLICE.
Binding to Occurrence	Publishers and subscribers bind to occurrence late, at runtime, as described in the Ice manual: IceStorm's default behavior maintains information about topics, links, and subscribers in a database. However, a message sent via IceStorm is not stored persistently, but rather is discarded as soon as it is delivered to the topic's current set of subscribers. If an error occurs during delivery to a subscriber, IceStorm does not queue messages for that subscriber [77, Sec. 44.3.7]. So both publishers and subscribers have to be available at the same time or an exception is generated (e.g., *ObjectNotExistException or NotRegisteredException)*.

4.3.3 Latency benchmarking

Performance is a quality attribute of middleware that can be measured in several different ways. First of all we might measure the time behaviour of the middleware and its resource utilization [47]. For middleware technologies time behaviour is often measured in terms of latency and throughput while the resource utilization is measured in terms of CPU consumption, bandwidth usage and memory consumption [48]. All the aforementioned metrics are important to understand the capabilities of the middleware and they are related. For example, evaluators will have to address trade-offs like latency versus bandwidth usage.

The objective of this benchmark is to show the implementation of a hands-on experiment to collect data for the evaluation of middleware as a part of PECA. I chose to evaluate performance as it is the most important quality attribute in the prioritization of the criteria (see Figure 4.2) and it is relatively easy to measure.

To narrow the experiment even further I only measured latency. The performance of all the communication protocol implementations should be measured depending on the requirements of the organization. I found that the latency of the one-to-one communication protocol implementations are especially interesting to evaluate because there are two CTA requirements which fulfillment are affected by this metric [4, Sec. 2.5]:

- **B-ACTL-2210**: Command delivery shall be reliable, i.e., commands must always be acknowledged. For actions requiring a significant amount of time, the system must display information that the action is on-going. The maximum time for acknowledgement or display of progress information is **2 seconds**.
- **B-ACTL-2280**: The delay between observations due to control software overhead must be **< 10 s**.

One-to-one communication protocols are used primarily for the control of devices (i.e., command delivery) so I measured latency as the the amount of time it takes to invoke a two-way operation on a server and receive the results of the operation [48], also known as Round-Trip-Time (RTT). The result might be an acknowledgement or return values.

Even though this is a fairly simple latency benchmarking, emphasis was placed on the correctness of the implementation. Practitioners often and unintentionally commit common mistakes when they implement their benchmarkings [78]. Below, the setup of the experiment and the results are presented. Section 5.5 provides some recommendations to avoid common mistakes in middleware benchmarking and how I tailored this particular experiment to avoid these mistakes.

Test setup

The test were executed in a closed and cabled client/server setup (see Figure 4.6) consisting in two hosts linked by a switch through an Ethernet connection. Even though this is a simple experiment which can be extended, it is still relevant due to the control hierarchy of ACTL described by Krugüer and Neppert [79] in which all the components can only be controlled by one component at the same time (control tree).

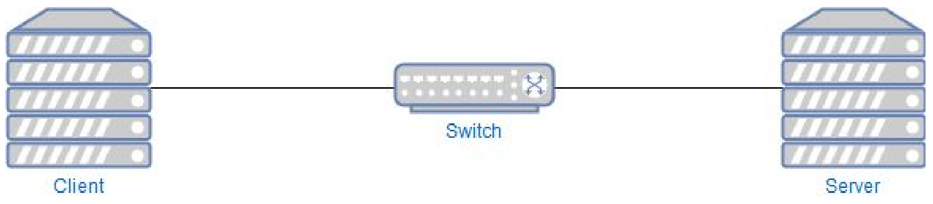


Figure 4.6: Setup of the benchmark.
Source: Elaborated by the author.

Both the client and the server application were executed in a separate HUAWEI FusionServer RH2288 V3. They supports up to two Intel® Xeon® E5-2600 v3 series processors. Each processor supports up to 18 cores. They have 24 memory slots 24 for DDR4 RDIMMs or LRDIMMs with a maximum memory capacity of 1.5 TB.

The ACTL team [15, Tab. 3.2] described the CTA topology, including the “ethernet lines connecting a telescope with the control building and the array-level trigger.” Specifically they propose a 1 Gbit/s Ethernet connection for slow control and a another one for the telescope drive system. Therefore I used a Huawei Quidway S5700-48TP-PWR-SI switch with a 1 Gbit/s Ethernet connection to link the two hosts.

Regarding the software, the test was performed using orVit 3.6 virtual machines with CentOS 7 operating system like the current implementations of ACS. The virtual machines were provided with 1 core and 1GB of RAM each.

In [79] the ACTL software architecture is described including the software components that are involved in the command delivery. Figure 4.7 shows a simplified ACTL command flow with the most representative components of the distributed system.

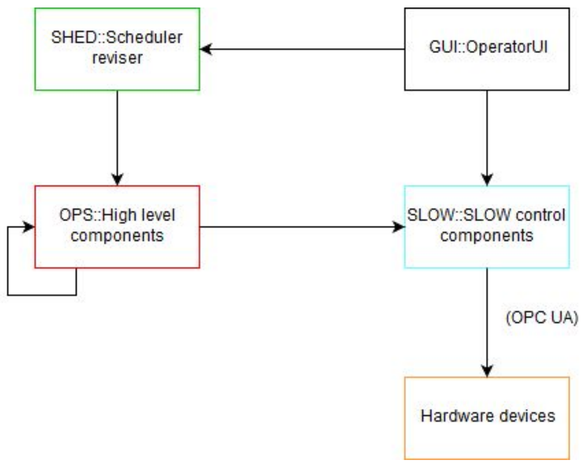


Figure 4.7: Simplified main command flow in ACTL.
Source: Elaborated by the author.

In a meeting with some members of the ACTL team I could obtain more updated information about the implementation of these components. The scheduler was programed using C++, the GUIs backend is Python based, OPS components are mainly made in Java and Python, and slow control components are implemented in Java and C++. The most representative scenarios (i.e., which are expected to occur at a higher frequency) are the following:

- A Java client controls a Java server.
- A Java client controls a C++ server.
- A Python client controls a Java server.
- A Python client controls a C++ server.

These scenarios are further confirmed by the current telescope prototypes such as the SST-1M in which the ACS control related components are built in Java or Python and C++ is used only for data acquisition [20]. Similarly in the ASTRI SST-M2 prototype almost all ACS components are in Java [80]. The graphical user interface prototype for the CTA operator also confirms our assumptions [81].

For testing these interactions the latest stable release of each middleware (at the moment of the experiment) was installed in the virtual machines:

- JacORB 3.9
- OmniOrb 4.2.2
- TAO 2.4.6
- ICE 3.7
- libzmq 4.2.3 (C++ and Python)
- jeromq 0.4.3 (Java)
- Protocol Buffers v3.5.0
- EPICS v4.6

In the meeting with some CTA experts we had the opportunity to talk about the general structure of the commands. ACS is currently based on CORBA, so it is not a surprise that commands are sent as a procedure with their arguments in RPC-like communications. More interesting are the types of arguments commonly sent in the ACTL control system, which are often basic types (e.g., integers, strings and floats) and data structures composed by those types. Nevertheless the commands are often light messages. For these reasons in this experiment the client-processes invoked procedures with an a structure as the only parameter. The structure itself contained variables of common native types (i.e., integer, single-precision floating-point, double-precision floating-point, Boolean and string).

Benchmarking results

In this section I show the results of the latency benchmarking. For clarity the results are expressed in terms of interactions (commands) per second. The reader might notice that there is not an ultimate best candidate for all the scenarios. Also for the interactions that involve Java the latency at runtime is around half the latency at warmup time (see Figures 4.8, 4.9 and 4.11) while in the only non-Java interaction the runtime and warmup latency are virtually the same (see Figure 4.10).

Ice has the lowest latency at runtime for the non-Python scenarios. During warmup it has an average speed when we compare it to the other candidates. On the other hand ZeroMQ with Protocol Buffers proved to be the slowest of all the candidates in every scenario at runtime and warmup. This might be explained by the fact that ZeroMQ was designed to be used for asynchronous communication and it relies in batched messages for increasing performance. Some tests were made to check that the bottleneck of performance was in ZeroMQ and not in the use of Protocol Buffers which is an easily replaceable serialization library. Nevertheless, in a synchronous use case ZeroMQ proved to be relatively very slow even while sending empty messages.

EPICS was the fastest at warmup in all Java related scenarios. Finally, JacORB seems to be slow if we compare it to OmniORBpy and TAO. The JacORB client to JacORB server interaction was the worst performance of CORBA. When we replace JacORB either in the client or the server side an improvement in latency is detected. CORBA has the best performance in the Python scenarios, completely out-matching its competitors with the OmniORBpy to TAO interaction. This shows how some ORB developers overcame some of the problems of the past and they were able to provide high performance ORBs.

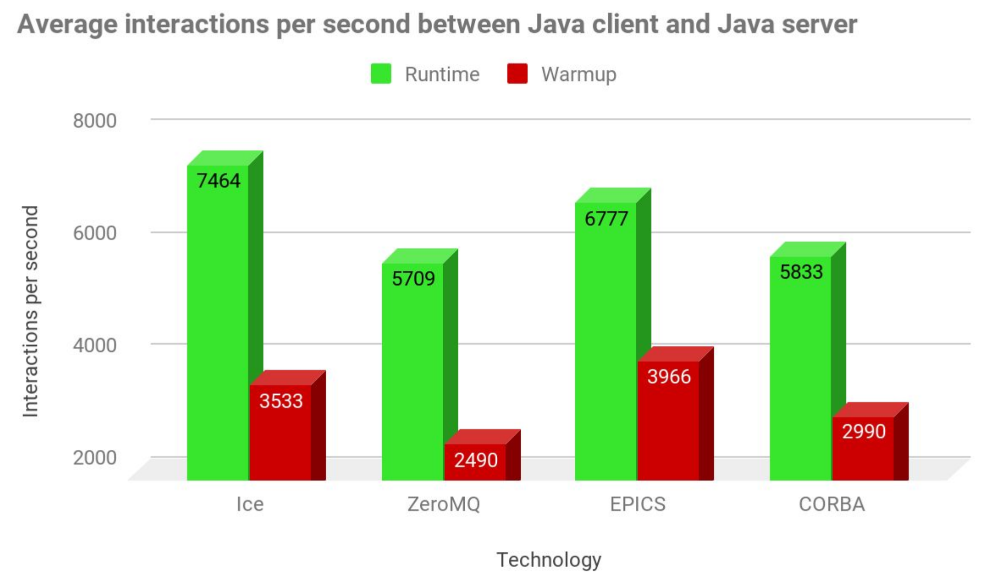


Figure 4.8: Average interactions per second between Java client and Java server.
Source: Elaborated by the author.

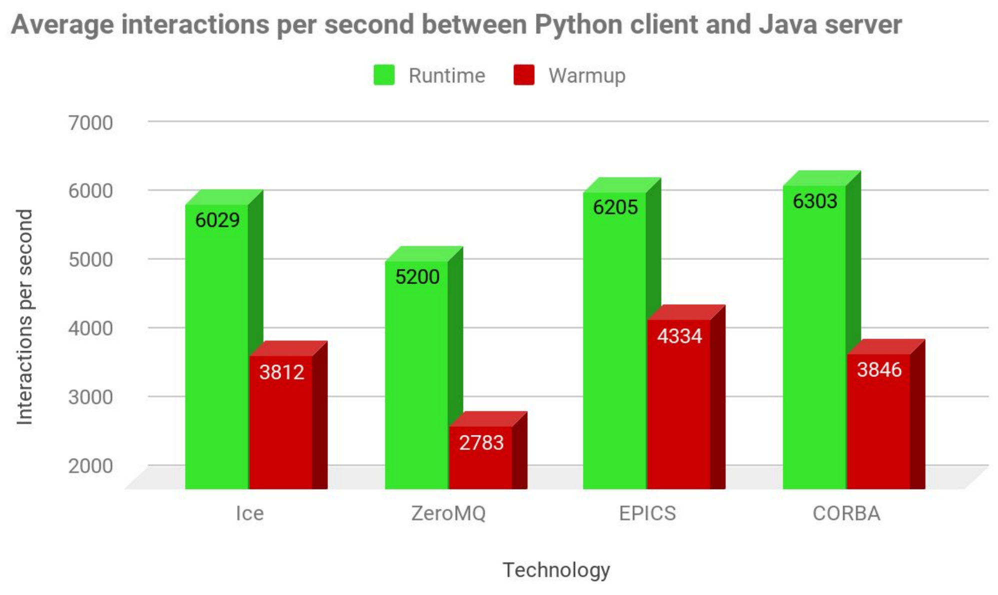


Figure 4.9: Average interactions per second between Python client and Java server.
Source: Elaborated by the author.

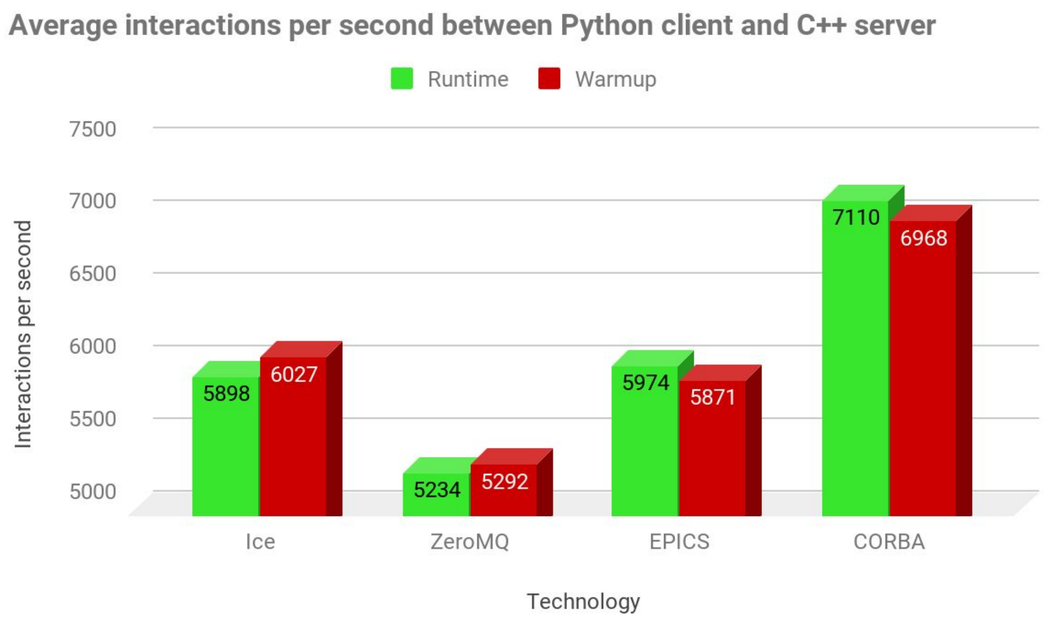


Figure 4.10: Average interactions per second between Python client and C++ server.
Source: Elaborated by the author.

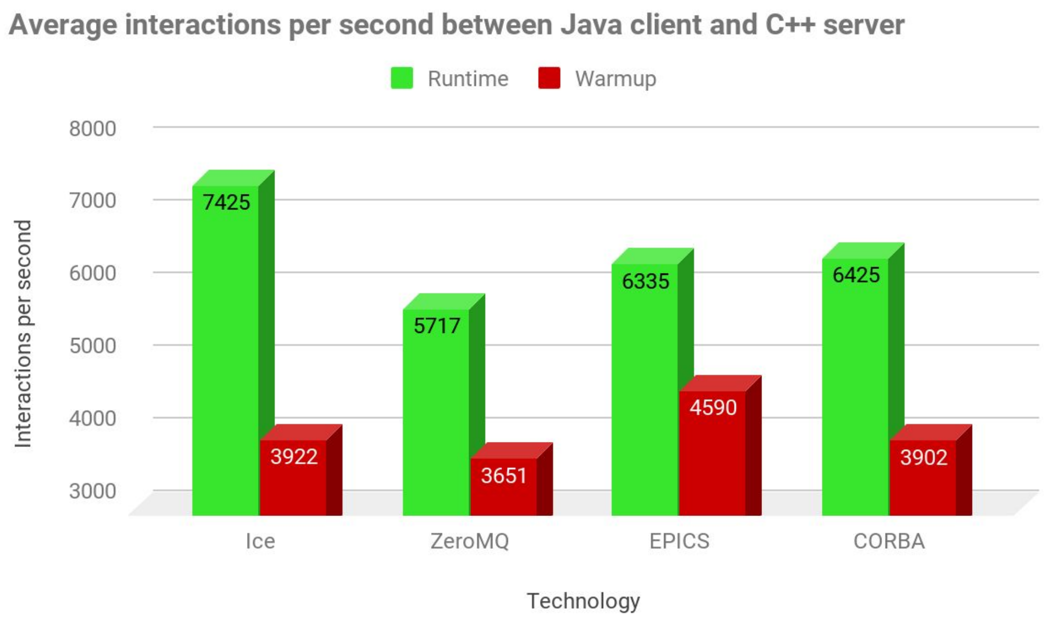


Figure 4.11: Average interactions per second between Java client and C++ server.
Source: Elaborated by the author.

4.4 Analysing the data

At this point PECA users have gathered a lot of quantitative and qualitative data. If this data is going to help the decisions makers to choose the more fitting technology then the first task in this step should be consolidate the data into useful information. The evaluators should be aware that after consolidating the data we lose some details in favor to easier understanding and “two very different products can appear to be virtually identical” in terms of fitness [16, Sec. 5.1]. Even though I followed a mathematical approach, the reader should understand that consolidating the data will not generate unquestionable facts (e.g., middleware A is better than B) because the procedure is based on data and judgment [16].

4.4.1 Picking an AHP approach

As pairwise comparisons from AHP were used to prioritize the evaluation criteria its seems natural to reuse the same technique to consolidate the data. The analytic hierarchy process provides two basic approaches to do this consolidation [36]:

- Absolute measurement: the evaluators use pairwise comparisons to create scales for each evaluation attribute (i.e., for each leaf node in Figure 4.2). Then the data collected about the candidates is translated using these new scales, generating a local score for each evaluation attribute.
- Relative measurement: instead of creating a scale for each criterion, the evaluators use pairwise comparisons to calculate de local scores of each candidate directly.

These local scores are a new representation of the quantitative and qualitative data collected in the previous step. After the local scores of the middleware candidates are generated AHP uses weighted aggregation to consolidate them into a global score of fitness in a ratio scale. Finally we can generate a ranking of the alternatives by only considering their global score.

Following the guidelines described in Section 5.6, I choose the relative measurement approach to consolidate the collected data. This approach is very similar to the prioritization of the criteria in Section 4.2.6. For each evaluation attribute the evaluators make pairwise comparisons about the alternatives by asking the following questions: “Which of the two candidates has the property or meets the criterion more? How much more?” [36, Sec. 1.3].

4.4.2 Consolidating the latency data

Understanding the data

In an attempt of making a relevant benchmarking I customized it to reflect the system while still maintaining its simplicity. If we go back to the *Collecting the data* step (see Section 4.3.3), we can see that even with a simple latency benchmarking several results can be obtained. I tested the latency of the communication between clients and servers implemented in three different languages (Java, Python and C++). For each relevant combination of client and server I measured the warmup and runtime latency independently. Which of those results should we use? One might argue that an average of all the results should be used to compare the candidates. Another one might say that, because Java is the main development language [79], only the Java client and server interactions should be used to compare the alternatives. Furthermore, Should we consider the warm up latency or the runtime latency?

Warmup performance is not very important if the clients and servers are long lived processes. The ACS components can be created at the beginning of the night and run for several hours until they are not longer needed for the observations. In this case warmup latency seems to be irrelevant. Yet in the scenario where a component fails the control and supervision hierarchy of ACTL should be able to create a replacement [79] which might be affected by reduced performance during the warmup phase. If ACS is reliable enough the components should be mainly running with runtime performance.

For simplicity I continued the evaluation process considering only the Java-to-Java scenario with runtime performance (see Figure 4.9) to show the consolidation of data. The benchmark results are included in Table 4.12.

Table 4.12: Latency at runtime between a Java client and a Java server.
Source: Elaborated by the author.

Technology	Interactions per second (Runtime)
Ice	7464
ZeroMQ	5709
EPICS	6777
CORBA	5833

Making pairwise comparisons

To make the pairwise comparisons the evaluators have to answer the previously mentioned questions: Which of the two candidates meets the latency criterion more? How much more? Due to the fact that latency is a quantifiable criterion measured using a ratio scale it may be tempting to emit the judgments by using the measured values directly to answer these questions like is shown in Table 4.13.

Even though this may appear to be objectively true, it is not. We should remember that fitness is always based on judgment. By reviewing the ACTL requirements, we might make better judgements about the fitness of each technology regarding the latency criterion [4, Sec. 2.5]:

- **B-ACTL-2210**: Command delivery shall be reliable, i.e., commands must always be acknowledged. For actions requiring a significant amount of time, the system must display information that the action is on-going. The maximum time for acknowledgement or display of progress information is 2 seconds.

- **B-ACTL-2280**: The delay between observations due to control software overhead must be < 10 s.

Table 4.13: Pairwise comparisons regarding latency using a ratio scale directly.
Source: Elaborated by the author.

PC #1	Ice	1.307	ZeroMQ	1
Ice latency: 7464 interactions per second.				
ZeroMQ latency: 5709 interactions per second.				
Ice meets the criterion 1.307 times better than ZeroMQ.				
PC #2	Ice	1.101	EPICS	1
Ice latency: 7464 interactions per second.				
EPICS latency: 6777 interaction per second.				
Ice meets the criterion 1.101 times better than EPICS.				
PC #3	Ice	1.280	CORBA	1
Ice latency: 7464 interactions per second.				
CORBA latency: 5833 interactions per second.				
Ice meets the criterion 1.280 times better than CORBA.				
PC #4	ZeroMQ	1	EPICS	1.187
ZeroMQ latency: 5709 interactions per second.				
EPICS latency: 6777 interaction per second.				
EPICS meets the criterion 1.187 times better than ZeroMQ.				
PC #5	ZeroMQ	1	CORBA	1.022
ZeroMQ latency: 5709 interactions per second.				
CORBA latency: 5833 interactions per second.				
CORBA meets the criterion 1.022 times better than ZeroMQ.				
PC #6	EPICS	1.162	CORBA	1
EPICS latency: 6777 interaction per second.				
CORBA latency: 5833 interactions per second.				
EPICS meets the criterion 1.162 times better than CORBA.				

To fulfill the first requirement the time for acknowledgement has to be equal or less than two seconds. We can break down the acknowledgement time into application time (when the system do application related work) and communication time (when the middleware transport the all the requests and responses required to execute a command). All four candidates can enable over 5000 request-response interactions per second which means that they need less than 0.0002 seconds for each interaction. Hypothetically, someone could argue that over 5000 interactions per second are good enough and incrementing the speed of the middleware further is not important for the system or it is much less important than optimizing the application code.

A similar thing happens with the second requirement. The control software overhead includes all the transport and executions of commands that are required for starting a new observation in the night. How many commands should be transported and executed between any two observations? This can only be answered with absolute certainty by executing the whole system using each candidate which is infeasible. There is a trade-off between a realistic test or experiment and its cost. Here is when expert knowledge comes in handy. Expert in distributed systems can emit judgements about how fit is each technology based on the results obtained in a limited latency benchmarking. Let's suppose that the experts of the team determine that any middleware that enable over 5000 interactions per seconds is excellent and if a candidate is faster than that it doesn't make any significant difference for the ACTL system. In this hypothetical scenario the judgments using the Saaty scale are described in Table 4.14.

Table 4.14: Pairwise comparisons regarding latency using expert knowledge (hypothetical case).
Source: Elaborated by the author.

PC #1	Ice	1	ZeroMQ	1
Ice and ZeroMQ enable over 5000 interactions per second. Ice is equally as preferable as ZeroMQ.				
PC #2	Ice	1	EPICS	1
Ice and EPICS enable over 5000 interactions per second. Ice is equally as preferable as EPICS.				
PC #3	Ice	1	CORBA	1
Ice and CORBA enable over 5000 interactions per second. Ice is equally as preferable as CORBA.				
PC #4	ZeroMQ	1	EPICS	1
ZeroMQ and EPICS enable over 5000 interactions per second. ZeroMQ is equally as preferable as EPICS.				
PC #5	ZeroMQ	1	CORBA	1
ZeroMQ and CORBA enable over 5000 interactions per second. ZeroMQ is equally as preferable as CORBA.				
PC #6	EPICS	1	CORBA	1
EPICS and CORBA enable over 5000 interactions per second. EPICS is equally as preferable as CORBA.				

Calculating the latency local scores

We can use the Super Decisions v3 software to process the judgments in both Table 4.13 and Table 4.14. The latency local scores generated by the software are shown in Table 4.15.

Table 4.15: Latency scores derived from different judgements.
Source: Elaborated by the author.

	Scores using quantifiable data directly		**Scores using the Saaty scale and expert knowledge (hypothetical)**	
Technology	**Normal mode**	**Ideal mode**	**Normal mode**	**Ideal mode**
Ice	0.28947	1.00000	0.25000	1.00000
ZeroMQ	0.22143	0.76496	0.25000	1.00000
EPICS	0.26288	0.90813	0.25000	1.00000
CORBA	0.22623	0.78152	0.25000	1.00000

The results in Table 4.15 prove that even if we follow a mathematical approach for generating scores, it is not an objective task. From the same data collected in the previous PECA step I generated different scores and both approaches have an inconsistency ratio of cero. This is why the evaluation team should not be composed by just the junior engineer. It also needs the support of senior engineers with intimate knowledge about the system which is especially true when we are dealing with complex distributed systems.

4.4.3 Consolidating the default coupling data

Using the quantifiable data directly to generate the local scores is often ill advised and should be previously validated as an adequate approach for comparing the candidates regarding a particular evaluation attribute. Moreover this approach is not feasible when we are dealing with intangible criteria and qualitative data. In this section I analyze the data about the coupling introduced to the distributed system by the different middleware alternatives.

Relationship between coupling and software dependencies

As was already described in Section 4.3.2 we can understand the concept of coupling as the amount of knowledge we need about one module to be able to understand another module [76]. Knowledge is not straightforwardly quantifiable and there is not a standard way to measure it in the context of distributed systems. Fortunately Wijegunaratne and Fernandez [50] present a practical framework for application architects and designers to assist them on making effective decisions regarding the system design and middleware selection. The framework points out which pieces of information are relevant to determine the coupling of a distributed system and, more importantly for us, the default coupling (i.e., coupling induced by the middleware).

Achieving the lowest possible coupling in distributed systems is generally considered a desirable goal because it is associated with a reduced development and maintenance cost, but this is not strictly correct. The way that the organizations implements their processes influences the design of software components, generating dependencies between them. These dependencies are “in the realm of the requirement” while coupling is a property of the software implementation that satisfies those requirements [50, p. 124]. Wijegunaratne and Fernandez describe two types of software dependencies [50]:

- Processing dependency
    - Simple processing dependency: when a component needs some tasks be performed remotely by another component to able continue and finish its own processing.
    - Transactional dependency: likewise simple processing, transactional dependency occurs when a component needs some tasks be performed remotely, but this time by a set of components. Furthermore, all these tasks are considered by the requester “a single logical unit of work, to be carried in an all or nothing fashion” [50, p. 117].
- Informational dependency: when a component needs to send some information to one or more remote components without expecting that it will be processed. This

typically occur in response to an event, changes in existing information or generation of new information.

Each of these dependencies can be implemented only by a set of suitable middleware interface types [50] which introduce a part of the default coupling in the system. This fact allows us to understand that there is an appropriate level of coupling associated to each type of software dependency and the goal of the implementation should be to achieve the lowest appropriate level. Using middleware interfaces with a lower than the appropriate coupling leads to integrity problems. Processing dependencies require interface types that introduce more coupling than informational dependencies. In a similar way implementing transactional dependencies require interfaces types with higher coupling than those required for implementing non-transactional dependencies [50].

Software dependencies of CTA

The way CTA defined their processes (e.g., scheduling and executing observations) is the source of the software dependencies required to implement these processes. After we find the software dependencies we can start to analyze if the middleware candidates provide the required interfaces and if the coupling of the system can be minimized with the available options to reach lowest appropriate level of coupling. Specifically, two types of software dependencies were detected:

- ACTL simple processing dependencies: they are primarily derived from the control system requirements. Specifically, for performing observations (e.g., a fixed time observation of a target), ACTL should be able to execute a sequence of actions. These actions are translated in a ordered sequence of commands that has to be executed remotely in the software components controlling the telescopes and auxiliary devices. The components orchestrating these sequences can't complete their work or continue their processing until all the commanded components do their work and acknowledge.
- ACTL informational dependencies: they are primarily derived from the monitoring system. Software components in charge of devices periodically send data to the monitoring system so it can be archived and used for other components such as the scheduler, the RTA and the operator UIs. This data might be alarms, errors, warnings or status data.

After identifying the software dependencies we have an idea of what is the appropriate level of coupling that we are looking for the implementation of each dependency by following the guidelines presented in [50]. Then, with the the data about the middleware interfaces that is collected in the third step of PECA, we can choose the best middleware interfaces to implement the software dependencies.

Summary of Ice default coupling data

Ice provides a *static invocation interface* which covers the simple processing dependency pretty well. With synchronous static invocations a serie of commands can be executed in order, enabling an easy passage of control. On the other hand Ice also provides *asynchronous static invocation* which can be used when commands require a considerable amount of time to be executed remotely and a blocked requester component is undesirable. Furthermore Ice provides a *dynamic invocation interface* which only partially implements dynamic communication (i.e., it doesn't support dynamic discovery of remote interfaces). The coupling-related data collected about these interfaces is summarized as below:

- Ice can enable an easy passage of control with the static invocation interface (desirable).
- Ice stores administrative information centrally in the infrastructure (middleware) by providing its own location service (i.e., IceGrid or IceDiscovery).
- Both interfaces together provide synchronous, asynchronous and static communication.
- The clients bind early to the form of the servers, at development time, with both static and dynamic invocation interfaces due to the use of the interface definition language SLICE. The clients never bind to the implementation of the servers because Ice provides several interoperable language bindings, nevertheless the dynamic invocation interface does not support Python. The Ice location services allow the clients to bind late to the occurence of a server, at runtime.

For satisfying the informational dependency Ice provides a Pub-Sub event distribution service: *IceStorm*. This service can receive data from monitored ACTL components and distribute it to the interested consumers. The coupling-related data collected about IceStorm is summarized as below:

- IceStorm makes the passage of control difficult (desirable).
- Ice stores administrative information centrally in the infrastructure (middleware) by providing its own location service (i.e., IceGrid or IceDiscovery).
- IceStorm provides asynchronous, available and static communication.
- The publishers using IceStorm bind early to the form of subscribers because they use a topic (SLICE) interface to agree about the message format at development time. The publishers never bind to the implementation of the subscribers because Ice provides several interoperable language bindings. By using the Ice location services, both publishers and subscribers can bind late to the occurrence of the IceStorm service, at runtime, but IceStorm does not persist messages so they have to be simultaneously available to be able to communicate.

Summary of ZeroMQ default coupling data

ZeroMQ provides the *Request-Reply pattern sockets* which can be used to fulfill the simple processing dependencies of ACTL. These sockets provide one-to-one communications needed by a client to request a server to do some job. The coupling-related data collected about these socket is summarized as below:

- ZeroMQ makes the passage of control difficult because it is an inherently asynchronous communication library (undesirable).
- By default ZeroMQ requires that the administrative information has to be hardcoded in the application. For this reason the information is distributed across the components of the distributed system.
- The Request-Reply pattern sockets provide asynchronous and static communication.
- The clients bind early to the form of the servers by using Protocol Buffers to agree about the message format, at development time. The clients never bind to the implementation of the servers because ZeroMQ and Protocol Buffers provide several interoperable language bindings. ZeroMQ doesn't provide any discovery service or location service out of the box, so the clients bind early to occurrence of the servers, at development time.

ZeroMQ arguably has been designed for satisfying informational dependencies with its *Publish-Subscribe pattern sockets*. Like most of the message oriented middleware products it provides great flexibility for many-to-many communications, but does not force the developers to use a message broker. The coupling-related data collected about these sockets is summarized as below:

- Publish-Subscribe pattern sockets make the passage of control difficult (desirable).
- By default ZeroMQ requires that the administrative information has to be hardcoded in the application. For this reason the information is distributed across the components of the distributed system.
- The Publish-Subscribe pattern sockets provide asynchronous, static, available (with proxies) and non-available (with message broker) communication.
- The publishers and subscribers using Protocol Buffers have to agree early about the message format, at development time. The publishers and subscribers never bind to the implementation of the proxies or the message broker, because ZeroMQ and Protocol Buffers provide several interoperable language bindings. ZeroMQ doesn't provide any discovery service or location service out of the box, so the publishers and subscribers bind early to occurrence of the proxies or the message broker, at development time.

Summary of EPICS default coupling data

EPICS provides synchronous and asynchronous *RPC interfaces* which can be used to satisfy the simple processing dependency of ACTL. Clients can invoke remote functionality provided in the form of services. The coupling-related data collected about these interfaces is summarized as below:

- EPICS enable an easy passage of control with its synchronous RPC interface (desirable).

- EPICS currently only support UDP broadcast as its name resolution method for the pvAccess protocol. This means that the administrative information is not centralized, but it can be discovered at runtime.
- Both RPC interfaces together provide synchronous, asynchronous and static communication.
- The clients bind early to the form of the servers, at development time because the service name and parameters have to be be known at development time. The clients never bind to the implementation of the servers because EPICS provide interoperable implementations for C++, Java and Python. The UDP broadcast mechanism allows the clients to bind late to the occurence of a server, at runtime.

For the informational dependencies of ACTL, EPICS provides a *Monitor interface* which implement a publish-subscribe pattern. The coupling-related data collected about this interface is summarized as below:

- The Monitor interface does not allow the passage of control (desirable).
- EPICS relies on UDP broadcast as its default name resolution method. This means that the administrative information is not centralized, but it can be discovered at runtime.
- The Monitoring interface provides asynchronous, available, static and dynamic communication.
- EPICS never binds to form, clients can retrieve and use the introspection interface of the server to create a Monitor at runtime. It never binds to implementation due to the compatibility between the Java and C++ implementation plus its python wrapper (pvaPy). EPICS binds to occurrence late at runtime when the client uses UDP broadcast to find the channels endpoints.

Summary of CORBA default coupling data

The CORBA implementations used by ACS provide three interfaces types that are suitable for satisfying the simple processing dependencies of ACTL: *static invocation*, *dynamic invocation* and *asynchronous method invocation*. The coupling-related data collected about these interfaces is summarized as below:

- CORBA enables easy passage of control with the static invocation interface (desirable).
- CORBA stores administrative information centrally in the infrastructure (middleware) by providing its own location service (i.e., CORBA Naming Service).
- The three interfaces together provide synchronous, asynchronous, static and dynamic communication.
- The clients can bind late to the form of the servers, at runtime, by using the dynamic invocation interface. The clients never bind to the implementation of the servers because JacORB, TAO and OmniORBpy can interoperate using IIOP protocol. The CORBA Naming Service allow the clients to bind late to the

occurence of a server, at runtime.

CORBA implementations provide a *Notification Service* which is suitable for satisfying the informational dependencies of ACTL. Each CORBA implementation considered in this evaluation provides its own notification service. ACS currently uses the TAO Notification Service. The coupling-related data collected about the CORBA Notification Service is summarized as below:

- The CORBA Notification Service makes the passage of control difficult (desirable).
- CORBA stores administrative information centrally in the infrastructure (middleware) by providing its own location service (i.e., CORBA Naming Service).
- The CORBA notification service provides asynchronous, available and static communication.
- The components rely on CORBA stub module generated by the IDL compiler to publish the data to the Notification Service so the binding to form occurs early at development time. They never bind to the implementation of the TAO Notification Service because it can interoperate with C++, Java and Python components. By using the CORBA Naming Service, both publishers and subscribers can bind late to the occurrence of the Notification Service, at runtime, but Neither TAO nor JacORB Notification Service implement the QoS properties required for guaranteed delivery so publishers and subscribers have to be simultaneously available to be able to communicate.

Making pairwise comparisons

By analyzing the data about the interfaces (summarized above) we are able to make pairwise comparisons between the middleware candidates regarding their default coupling using the Saaty scale. Due to the high amount of data considered in each judgment I show the judgments rationale in a very condensed format in Table 4.16.

Table 4.16: Pairwise comparisons regarding default coupling using the Fundamental scale.
Source: Elaborated by the author.

Pairwise comparison #1	Ice (6)	ZeroMQ (1)
	Simple processing dependencies	
Communication types	🟩 Synchronous and asynchronous	🟥 Only asynchronous
Administrative info.	🟩 Centralized in the middleware	🟥 Distributed in the application
Binding to occurrence	🟩 During runtime	🟥 During development time
	Informational dependencies	
Communication types	🟥 Available	🟩 Unavailable (with message broker)
Administrative info.	🟩 Centralized in the middleware	🟥 Distributed in the application
Binding to occurrence	🟩 During runtime	🟥 During development time
Ice is strongly to very strongly more preferable than ZeroMQ regarding default coupling.		

Table 4.16: Continued

Pairwise comparison #2	Ice (3)	EPICS (1)
Simple processing dependencies		
Communication types	Dynamic partially supported	Only static
Administrative info.	Centralized in the middleware	*Distributed in the middleware*
Informational dependencies		
Communication types	Only Static	Dynamic and static
Administrative info.	Centralized in the middleware	*Distributed in the middleware*
Binding to form	During development time	During runtime
Ice is moderately more preferable than EPICS regarding default coupling.		
Pairwise comparison #3	Ice (1)	CORBA (1)
Simple processing dependencies		
Communication types	Dynamic partially supported	Dynamic fully supported (but complex)
Ice is equally preferable as CORBA regarding default coupling.		
Pairwise comparison #4	ZeroMQ (1)	EPICS (3)
Simple processing dependencies		
Communication types	Only asynchronous	Synchronous and asynchronous
Administrative info.	Distributed in the application	*Distributed in the middleware*
Binding to occurrence	During development time	During runtime
Informational dependencies		
Communication types	Unavailable and static	Available and dynamic
Administrative info.	Distributed in the application	*Distributed in the middleware*
Binding to form	During development time	During runtime
Binding to occurrence	During development time	During runtime
EPICS is moderately more preferable than ZeroMQ regarding default coupling		
Pairwise comparison #5	ZeroMQ (1)	CORBA (6)
Simple processing dependencies		
Communication types	Only asynchronous and static	Synchronous, asynchronous, static, dynamic
Administrative info.	Distributed in the application	Centralized in the middleware
Binding to occurrence	During development time	During runtime
Informational dependencies		
Communication types	Unavailable (with message broker)	Available
Administrative info.	Distributed in the application	Centralized in the middleware
Binding to occurrence	During development time	During runtime
CORBA is strongly to very strongly more preferable than ZeroMQ regarding default coupling.		
Pairwise comparison #6	EPICS (1)	CORBA (3)
Simple processing dependencies		
Communication types	Only static	Static and dynamic
Administrative info.	*Distributed in the middleware*	Centralized in the middleware
Binding to form	During development time	During runtime
Informational dependencies		
Communication types	Dynamic and static	Only Static
Administrative info.	*Distributed in the middleware*	Centralized in the middleware
Binding to form	During runtime	During development time
CORBA is moderately more preferable than EPICS regarding default coupling.		

Generating the default coupling local scores

By introducing the judgments of the pairwise comparisons in an AHP software package we can generate local scores for each candidate regarding its default coupling which then can be easily aggregated with the rest (e.g., with the latency local scores). Table 4.17 shows the local scores in normal mode and ideal mode. These scores were generated using the software Super Decisions v3.

Table 4.17: Default coupling local scores derived from judgement.
Source: Elaborated by the author.

	Scores using the Saaty scale and expert knowledge	
Technology	**Normal mode**	**Ideal mode**
Ice	0.39647	1.00000
ZeroMQ	0.05998	0.15129
EPICS	0.14708	0.37096
CORBA	0.39647	1.00000

4.4.4 Generating the global scores (weighted aggregation)

Once the local scores of all the evaluation attributes are calculated, it is time to consolidate them using weighted aggregation. The global score of each candidate is the sum of each evaluation attribute local score multiplied by its global priority. The global priority of an evaluation attribute is obtained by weighting (multiplying) its local priority by the priority of all its ancestors in the tree representation of the hierarchy. The local priorities were obtained in the *Establishing the criteria* step of PECA.

Nonetheless I only collected data about two evaluation attributes. Without also collecting the required data about the other 42 evaluation attributes we are unable to use the model shown in Figure 4.2 to generate the global scores of the middleware alternatives. For this reason I propose an oversimplified model presented in Figure 4.12 to show the reader how to properly consolidate quantitative (latency) and qualitative (default coupling) data. The priorities of the criteria in this model are obtained by reusing the paired comparison between *Product Quality Characteristic* and *Strategic Concerns* from Table 4.8.

I also made a Super Decisions v3 model for this simplified decomposition. Figure 4.12 shows the local priorities for each criterion. In each level the sum of the local priorities of the criteria is 1. The global priorities for Latency and Default Coupling are the following:

- Latency global priority = 0.25000 * 1.00000 = 0.25000
- Default Coupling global priority = 0.75000 * 1.00000 = 0.75000

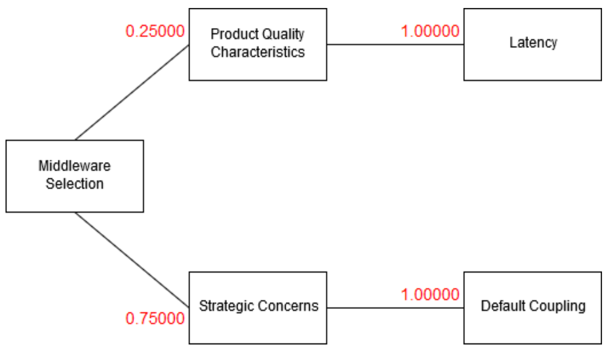


Figure 4.12: Simplification of the full hierarchical decomposition.
Source: Elaborated by the author.

AHP describe two modes for synthesizing the model. What mode should we use? The same problem with the same criteria might be approached with both modes, it only depends on what the evaluators are trying to accomplish. If the goal is to choose the middleware that "stands out" (i.e., dominance is important) then the normal mode (also known as distributive mode) should be used [51, Sec. 2.7.1]. If the goal is to choose "a well performing" middleware (i.e., performance is important) then the ideal mode is the best choice [51, Sec. 2.7.1]. Therefore I used the ideal mode in this evaluation. In [82] more detailed guidelines for choosing the synthesis mode are provided. Table 4.18 presents the global score of each candidate for our simplified hierarchy. Table 4.19 shows the ranking of the candidates based on this partial evaluation. The reader should remember that only 2 of the 44 evaluation attributes were measured so these global scores must not be used for real decision making.

Table 4.18: Global priorities of the candidates (simplified problem).
Source: Elaborated by the author.

Technology	**Global Score**
Ice	0.25000 * 1.00000 + 0.75000 * 1.00000 = 1.0000000000
ZeroMQ	0.25000 * 0.76496 + 0.75000 * 0.15129 = 0.3047064131
EPICS	0.25000 * 0.90813 + 0.75000 * 0.37096 = 0.5052543648
CORBA	0.25000 * 0.78152 + 0.75000 * 1.00000 = 0.9453800221

Table 4.19: Partial ranking of the candidates.
Source: Elaborated by the author.

Place	**Technology**
First	Ice
Second	CORBA
Third	EPICS
Fourth	ZeroMQ

I would like to note that Super Decisions v3, for an unknown reason, does not implement the ideal mode synthesis as described in this section so the users must manually calculate the scores as shown in Table 4.18. There are several specialized packages for supporting AHP such as Expert Choice[3], Logical Decisions[4] and Make it Rational[5] [83]. Almost all AHP packages are commercial software, so I could not test them for this dissertation. The ones with free trials were too limited. Even the classic software for supporting AHP, Expert Choice, has some errors in its implementation of the ideal mode [82]. I finally choose Super Decisions v3 because it is free, with good documentation and it is very complete regarding its features. If the reader remains unconvinced about using Super Decisions v3 then he or she can always use a spreadsheet to do all the necessary calculations for AHP.

In Section 4.2.6 I used the pairwise comparison technique for calculating the priorities of the criteria. In this section I used it again for consolidating heterogeneous data. The guidelines of group decisions-making and conflict resolution of Section 5.3 also apply here. If the reader is interested in this topic I encourage he or she to check the details in Chapter 8 and Chapter 9 of [84].

4.4.5 Analyse the consolidated data

Comella-Dorda et al. [16, Sec. 5.3.1] describe the analysis as a "creative task" that requires "simply sound and careful reasoning". In this section I use some techniques recommended by them that may help us to analyze the data that was consolidate into useful information in a more effective way.

Sensitivity analysis

The sensitivity analysis shows how potential changes on the priorities generated in previous steps, specially the priorities of the evaluation criteria, might alter the global scores of the middleware candidates. This technique allows the evaluators to determine the robustness of the ranking obtained in Section 4.4.4 and the criteria that drove the ranking [84]. Furthermore, by executing a sensitivity analysis the evaluators can handle the conflicting interests of stakeholders, changes in system requirements, changes in maintenance strategies and potential expert bias [16].

Super Decisions v3 only supports sensitivity analysis for the distributive mode synthesis. A workaround for this problem might be to use the copy to clipboard function to collect the ideal local score of each evaluation attribute and paste them on a spreadsheet. I used the free Google Sheet application[6] for making my own sensitivity analysis. Figure 4.13 shows how the global score of each middleware alternative varies when we adjust the priority of the Product Quality Characteristic criterion.

[3] https://expertchoice.com/
[4] http://www.logicaldecisionsshop.com/catalog/
[5] http://makeitrational.com/pricing
[6] https://www.google.com/sheets/about/

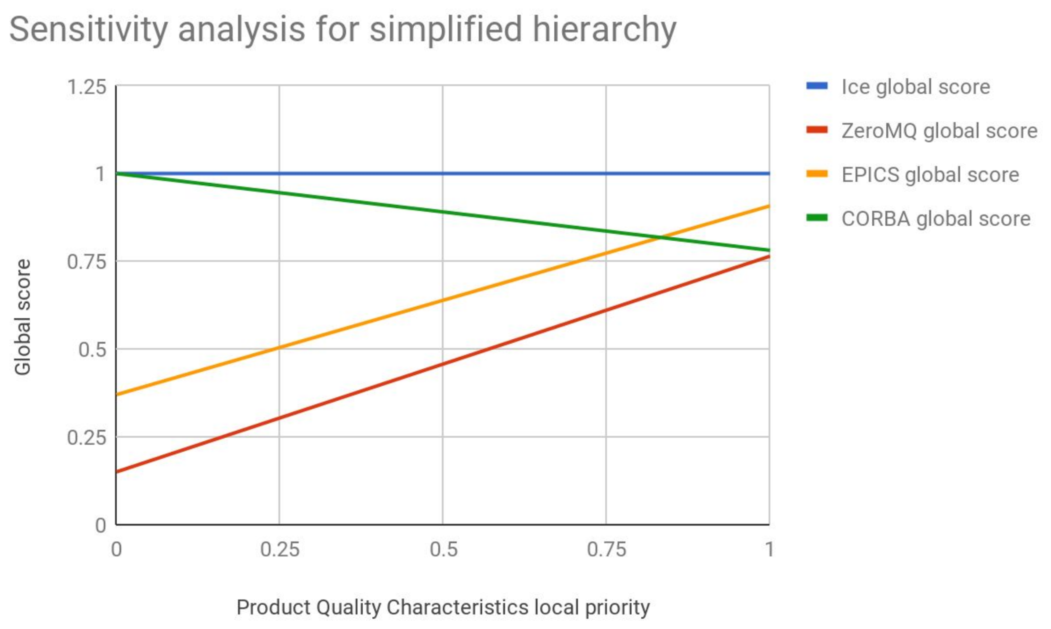


Figure 4.13: Sensitivity analysis for the simplified hierarchy.
Source: Elaborated by the author.

As the reader might notice Ice has a constant global score of 1 (the best possible score) regardless the priorities of the criteria. This only happens in the exceptional case when a candidate is the best regarding every criteria. Now that we have a visual representation, What should we do with it? Several questions can be answered by a sensitivity analysis. For example in [84] the author propose two:

- What happens when all the criteria have the same weight?
- What weight is needed for the Product Quality Characteristics criterion to lead to a tie in the overall scores of the alternatives?

First of all we note that the original ranking (see Table 4.19) for our test case is shown in Figure 4.13 when the Product Quality Characteristics priority is left unchanged (0.25). This ranking is preserved when we adjust it to 0.5 (i.e., when the two criteria have the same weight). Moreover Ice is the the preferred alternative regardless the prioritization of the criteria. CORBA can tie the score of Ice in the hypothetical case in which *Product Quality Characteristics*, which I simplistically measured as latency, do not matter (i.e., its local priority is 0). When the importance of the performance of the middleware start to increase the global score of CORBA start to decrease to the point in which EPICS surpases CORBA in the ranking and it position itself at second place, when the priority of the Product Quality Characteristic is around 0.84.

From this analysis we can say that, for my oversimplified test case represented in Figure 4.12, Ice is the clear best choice because, regardless the stakeholders interests, potential changes on system requirements (and maintenance strategies) or expert bias in the prioritization of the criteria, Ice always has the higher global score. We arrive to this conclusion because, in order to be able to show the execution of PECA, I reduced the complexity of the problem too much (i.e., only two evaluation attributes were considered). It is to be expected that when we use a hierarchy that represent the real system, like the one I proposed in Figure 4.2, the analysis will be more complex and accurate.

So far I only introduced the sensitivity analysis for the criteria. In some rare cases the evaluators can also do a sensitivity analysis about the local scores of the alternatives regarding certain evaluation attributes. This can be especially useful when the evaluators believe that the data collected for certain attributes is not very trustworthy or it has low quality (e.g., latency results from a benchmarking conducted by vendor). In this iteration a sensitivity analysis about latency may be useful to also consider the results of the scenarios that were discarded in the consolidation of the latency data (see Section 4.4.2).

Gap analysis

This technique organize the data obtained about each criterion in a visual representation that enables the evaluators to do a more straightforward analysis by comparing the candidates gaps to fulfill the criteria. This representation consist in a matrix whose cells describe the gap (see Table 4.20).

Table 4.20: Template of gap analysis matrix for four candidates and two criteria.
Source: [16].

Criteria	**Products**			
	P1	P2	P3	P4
C1				
C2				

We might describe the gaps in different ways. For instance the evaluators can use Boolean values (i.e., true or false) to fill the matrix or use the local scores of the evaluation attributes that were obtained in Section 4.4.2 and Section 4.4.3. The most useful way to build the gap analysis matrix is to fill the cells with a description of the respective gap so then we can determine the cost of fulfillment of that gap more easily.

Table 4.21 shows a gap analysis matrix with only default coupling as a criterion. As I mentioned in Section 4.4.3, this criterion is related to the need of building and maintain a complex distributed system with limited manpower. To accomplish this it is desirable that the middleware introduce the lowest appropriate coupling to the system, so the gaps described in the matrix are the missing features that make the default coupling of the middleware candidates diverge from the appropriate level. “Overabundance of features” can also generate a gap when it is undesirable to expose these additional characteristics to certain users and the features “must be hidden” [16, Sec. 5.3.3].

Table 4.21: Example of a gap analysis matrix with only one criterion.
Source: Adapted from [16].

Criteria	**Products**			
	Ice	ZeroMQ	EPICS	CORBA
Default coupling	Limited dynamic communication support No unavailable communication (Icestorm)	No synchronous communication Administrative information at application level Distributed administrative information No dynamic communication	Distributed administrative information No dynamic communication (RPC) No unavailable communication (Monitor interface)	No dynamic communication (Event service) No unavailable communication. (Event service)

In this PECA iteration all the products had a gap to be filled regarding their default coupling, but this is not always the case. Some middleware candidates might be a complete solution regarding some criteria so their cost of fulfillment for those criteria is zero.

Cost of fulfillment

After generating the gap matrix with all the relevant criteria the evaluators have a good idea of what is missing to fulfill the requirements of the system if they choose any one of the candidates. Now it is time to estimate how much will it cost to close all the gaps for each product. The cost might be expressed in various forms such as dollars, time or shifted risk. The approach proposed by Comella-Dorda et al. [16, Sec. 5.3.3] for determining the cost of fulfillment for each candidate has the following steps:

- Step 1: Identify each gap.
- Step 2: For each gap, determine
    - one or more fulfillment strategies (e.g., negotiate with the vendor, modify other system components, add custom code, negotiate requirements)
    - the cost to implement each strategy (costs include architecture/design, maintenance, process changes, etc.)

- Step 3: Select the preferred set of fulfillment strategies.

Once the previous steps are done, the evaluators can obtain the total cost of fulfillment associated to each candidate and then select the one with the lowest cost. The execution of this technique requires a high level of expertise that evaluation teams often do not have. Consequently the evaluators have to team with experts on architecture, design, maintenance and business for determining the fulfillment strategies and their respective costs [16]. Even though I did not have access to these experts, for completeness, I proposed some fulfillment strategies for ZeroMQ:

- No synchronous communication:
    - Custom implementation of a synchronous communication protocol by the ACTL team.
    - Integrate ZeroMQ with a middleware product that provides an efficient synchronous protocol implementation.
    - Use ZeroMQ as it is and manage the risk of having integrity problems during the development process.
- Administrative information distributed at application level:
    - Custom implementation of a location service to hold the administrative information.
    - Integrate ZeroMQ with a technology that already solved this problem (e.g., Ice and CORBA).
    - Use helper libraries such as ZeroConf [63].
    - Implement an UDP-based discovery protocol [63].
- No dynamic communication:
    - Use Protocol Buffers self-describing messages technique (only for C++ and Java).
    - Replace Protocol Buffers with another serialization library such as MessagePack or use JSON when dynamic communication is needed.

Each of these strategies has an associated cost which has to be estimated. Evaluators should be careful with choosing incompatible fulfillment strategies for different criteria regarding the same middleware candidate [16].

4.4.6 Making recommendations

As explained by Comella-Dorda et al. [16, Sec. 5.4] this task is about providing all the necessary information to the decision makers so they are able to “make an informed decision.” Even when the evaluators can’t recommend any candidate, they have gained through the evaluation process knowledge and understanding of the system that should be documented. This knowledge might be used to support architecting, wrapping, testing and maintenance [16].

The three documents that serve as an output for this iteration of PECA and as an input for the next one are the Product Dossier, the Evaluation Record and the Summary with Recommendations whose templates are described in Appendix B, Appendix C and Appendix D respectively. A detailed description of how to build each document can be found in [16]. Documentation is often considered a tedious task that practitioners try to avoid, but this should not be the case. By checking the templates the reader might notice that, at this point, most of the work has been already done.

In regard to the Product Dossier, it is not necessary to include the full information of all candidates considered in the evaluation. During the quick assessment I gathered information about the coarse filters and discarded 12 candidates (see Section 4.3.1). A summary of this information with a brief explanation about the rejection of these 12 middleware alternatives is enough as a dossier for these candidates. Similarly the first and the last section of the Evaluation Record can be generated using the information in Section 4.1. The criteria record can be filled with the criteria generated in Section 4.2. Since the hierarchical decomposition method described in [35] was used, I found more intuitive to structure the criteria record as is shown in Table 4.6 for performance criterion. The information for the result record was generated in Section 4.4.4 and Section 4.4.5.

The final task of PECA is to make and document the Summary of Recommendations (see Appendix D). The analysis of fitness consolidates the information generated during the whole evaluation process. Regarding the evaluation deficiencies of the current effort it should be obvious at this point that a new iteration of PECA, executed by the ACTL team, is necessary for selecting the most suitable middleware candidate. Even though I am confident about the results of this iteration regarding the correctness of the implementation which is key for accomplishing the objective of this dissertation I insist that, because it was just a partial execution of the process, the results (e.g., the ranking shown in Table 4.19) must not be used directly for the decision making (i.e., the middleware selection). I generated a full hierarchical representation of the decision problem for ACTL, but then only 2 from the 44 evaluation attributes were considered due to the scope and objectives of this iteration. Furthermore I was unable to use techniques for collecting data about external metrics (see Section 5.4) due to budget and infrastructure constraints. During the execution of PECA the evaluation team is very likely to discover some unexpected facts. For instance some discovered facts in this iteration are the following:

- ZeroMQ is not the best option for synchronous command execution.
- Ice dynamic invocation only partially supports dynamic communication.
- EPICS UDP broadcast for service discovery may conflict with firewalls.
- Non-Java CORBA implementations are faster than some people might think.
- The criterion *Multi-language Interoperability* obtained a relatively low priority in this first iteration of PECA after the analysis of the CTA documentation (see Figure 4.2). Nevertheless in a meeting with some CTA experts I learned that the development teams working in each subsystem of ACTL have strong preferences for different programming languages, effectively seizing the multi-language interoperability of ACS.

These documents are just a way of storing the knowledge generated during the evaluation process in a convenient way so it can be accessed in the future. For example, if after choosing a product the development team find problems during its integration with the system, they can review the rationale behind the selection of the product and alternatives strategies for the use of the product. If the problem couldn't be solved, then the evaluation team might start new iteration of PECA which is expected to be faster, cheaper and overly more effective than the previous one. To this end the evaluators are encouraged to tailor the templates of these documents for preserving and transmitting their improved understanding of the system in the most effective way possible.

While no meaningful recommendation for selection can be made by analyzing less than 5% of the evaluation attributes, I would like to point out ZeroC Ice as a promising candidate for further evaluation in a forthcoming iteration of PECA. The rationale for supporting Ice includes the following points:

- Ice is similar to what already exist. It implements the same communication paradigm as CORBA. This might be translated into reduced architectural mismatches and easier learning curve for people already familiar with ACS.
- According to my custom latency benchmarking, the Java binding of Ice is fastest and Java is arguably the most important programing language for the ACTL team.
- Ice can effectively reduce the development and maintenance effort by introducing low coupling, providing network transparency and a suit of standard services which may be used to ease the development of the ACTL system.
- Ice introduces only a little more coupling than CORBA into the system, but it appears to bring in much less complexity with its simpler API.
- Its documentation and support are very good.
- It appears to have an active community and a mature codebase.
- It can support the high availability and reliability required by ACTL by increasing the overall fault tolerance of the system with physical redundancy mechanisms and time redundancy mechanisms. Ice can also replicate its own standard services.

Nonetheless the previous points must be backed up with quality data relevant to the system context. I took a first step on this direction by measuring latency and default coupling, but there is much work to do. For this reason I can only recommend to continue the evaluation.

As was repeatedly explained in this document, the fitness of the software is partially based on judgment which is not purely objective. This also applies to the generation of the model to represent the problem. The interested parties might use the full hierarchy of Figure 4.2 as a starting point, and modify it to reflect the real context of their system. I built the full hierarchical representation with ACTL needs in mind so CTA can reuse it with less modifications than any other organization trying to solve the same problem.

CHAPTER 5: SUMMARY OF ADVICES AND RECOMMENDATIONS

This chapter includes several tips that I compiled from the literature that are important or were especially helpful while executing the evaluation and selection process.

5.1 Forming a good evaluation team

The process described in this dissertation requires an evaluation team with a large range of skills [16]. This is especially true when we are dealing with complex distributed systems. Specifically software architects, designers, developers, maintainers, business analysts and end users are required.

Furthermore, the evaluation team should include some experts on the topics mentioned above, because junior engineers lack the expertise required to complete certain task of the process such as creating a polished hierarchical representation of the MCDM problem, consolidating quantitative data with qualitative data and calculating the *cost of fulfillment* for each middleware candidate.

Finally, the evaluation team should avoid to bias the results at all cost. To accomplish this a balance of power is required [16] so no individual member is able to consciously or unconsciously manipulate the results of the evaluation.

5.2. Choosing the adequate criteria and metrics

Even though I used the hierarchical decomposition process to derive the evaluation attributes from the from the influencing factors it is important to be conscious about the fact that the criteria decomposition is by definition a subjective process based in the knowledge and expertise of the evaluators. We can find the decomposition of some common criteria in the literature, but no general list of criteria is adequate for all evaluation contexts and trying to blindly follow one will probably introduce errors of inclusion and exclusion in the evaluation process. An example of these decompositions are the ISO and IEEE standards about software quality [47].

Another concern is the choice of metrics for the evaluation criteria. The evaluation attributes are the criteria that can be measured using some kind of metric to evaluate the different alternatives of communication middleware. Even though we can measure *performance* relying only on quantifiable metrics, this is not always the case. For instance let's consider the *middleware support to the system extensibility*. The evaluation attribute (*default coupling*) is a description about the type of interfaces provided by the middleware [50]. This is all right because we are dealing with a multi-criteria decision making problem in which intangible criteria might be (and often are) more important than quantifiable evaluation criteria and thus they have to be included.

Finally regarding the metrics an important lesson was learned from [47] about the different types of metrics. The ISO standard classify metrics in three types:

- Internal metrics: we use these metrics to measure the intrinsic properties of non-executable or intermediate software products (e.g., documentation, source code, etc) during the design and coding phases of the software development lifecycle. This includes the properties “which can be derived from simulated behaviour” [47, p. 15].
- External metrics: they allow to measure the software product quality indirectly by measuring the behaviour of the system that contains it. We can use these metrics in simulated environments with simulated data during the testing phases of the software development lifecycle.
- Quality in use metrics: we use these metrics to “measure the extent to which a product meets the needs of specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in a specified context of use” [47, p. 15]. There may be different types of users with different perspectives which have to be addressed (e.g., operators and maintainers).

One can argue that using quality in use metrics is the ultimate way to determine if the software system succeeded. Let’s consider the following requirement of ACTL :

- “**A-RAMS-0060**: the availability of the Array Control Software, including Data Acquisition, must be >99.5 %. *Applicable States: Observing*” [4, Sec. 2.1.2].

Ideally we would want to measure the middleware and its impact on the system availability when the middleware is already fully integrated in the real system and it is executed by real users in a real context of use. By using quality in use metrics (e.g., safety and satisfaction metrics) evaluators could determine with almost no uncertainty if the system fulfilled the requirements of the users.

Yet we evaluate software to reduce the risk of the system failing in the future. Therefore the problem with quality in use metrics is that when we are able use them it is already too late, because the cost required to integrate a communication middleware in the system is too high. Fortunately for us, the three types of metrics mentioned above are correlated as shown in Figure 5.1 [47].

External metrics can be used to predict the value of the quality in use metrics by allowing to evaluate the software product during testing and operation in a simulated environment. Internal metrics can be used to predict the value of the external metrics by allowing to evaluate the software product early, even before the software product becomes executable [47].

CTA can use external metrics in an iteration of PECA due to the fact that they already have the appropriate infrastructure such as sophisticated test beds, test harnesses and prototypes. Some particularly valuable external metrics that I recommend for this evaluation are described in Table 5.1 and Table 5.2.

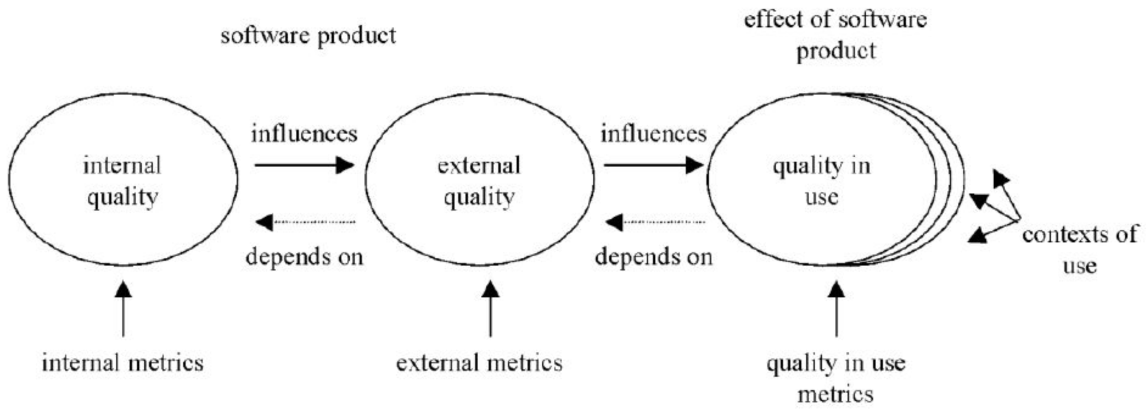


Figure 5.1: Relationship between types of metrics.
Source: [47]

Unfortunately in this iteration of PECA I didn't have the resources to use external metrics directly so I had to rely on internal metrics instead. In Section 4.3.2 and Section 4.3.3 I show how to use collecting data techniques for measuring two internal metrics: default coupling and latency.

Table 5.1: Availability as an external metric in ISO 9126-1.
Source: Elaborated by the author

Name	Availability
Description	The readiness of the system for use during a specified period of time
Measurement	Operation time / (operation time + mean time to repair)
Method of measurement	Test the system in a product like environment for a specified period of time performing all user operations. Measure the repair time period each time the system was unavailable during the trial. Compute the mean time to repair.

Table 5.2: MTBF as an external metric in ISO 9126-1.
Source: Elaborated by the author

Name	Mean time between failure
Description	The frequency in which the software fail in the operation
Measurement	Operation time / total number of actually detected failures
Method of measurement	Count the numbers of failures occurred during a defined period of operation and compute the average interval between the failures.

It might be tempting to reuse evaluation criteria directly from standards, literature or from previous evaluations. For example, the CERN conducted a middleware evaluation for the Controls Middleware (CMW) project and it presented some of the details in [10, 57]. In these documents a set of requirements and criteria are listed. Even though the contexts of the systems might be similar (i.e., CERN is trying to replace CORBA in their control system) the evaluators executing PECA should be careful and refrain from copying these criteria without previously considering their significance to their own system.

Finally if the reader is having doubts about the correctness of his or her evaluation criteria it is useful to remember the four characteristics of good criteria described by Comella-Dorda et al. [16]:

- Assessability: data can be gathered to measure the alternatives regarding the criterion.
- Ability to discriminate: the criterion has to be useful to discriminate which alternative is better.
- Non-overlapping: some overlapping criteria can be tolerated but it is not desirable because It might generate misleading results and it also affects negatively the use of AHP techniques.
- Significance: the criterion has to be valuable in the context of the system.

These principles can help the evaluators to refine the hierarchy. For example I included the *bulk data transfer support* evaluation attribute as a child of the *services support* node in the hierarchical representation of the problem. This evaluation attribute is assessable through a performance benchmark, it has the ability to discriminate quantitatively between the candidates and it is significant because if a middleware doesn't provide support for bulk data transfer then it will have to coexist with other middleware (e.g., ZeroMQ) or a protocol implemented by the ACTL team (maybe based on TCP) increasing the integration cost and development cost. Nonetheless *bulk data transfer support* overlaps with *throughput* so I removed the former from the hierarchy (see Figure 4.2).

5.3 Group decision-making and conflict resolution

Even though group decision-making and conflict resolution is out of the scope of this dissertation, it is likely that CTA or any similar organization that might use this guide will face these problems. In this section I briefly summarize the solutions proposed in [84].

Group decision-making is important because it allows the evaluators to consider the expectations of different stakeholders and incorporate the knowledge of the experts in different areas of this multiple-criteria decision-making problem. The solution proposed in [84] consist in (optionally) dividing the full hierarchy representing the problem in sub-hierarchies and then assigning the task of prioritizing each of those sub-hierarchies to different subcommittees of experts. These subcommittees can work independently of each other. For example, we might split the hierarchy described in Figure 4.2 into four and assign the task of prioritizing the sub-hierarchies of *Functional Requirements* and the *Product Quality Characteristic* to one subcommittee (because these topics are very related) while another subcommittee might work on the *Strategic Concerns* and *Domain and Architecture Compatibility* sub-hierarchies. It is expected that a group of experts will participate in each subcommittee so there has to be a way of aggregating the judgments of all the participants working together. The standard way of aggregating judgments is by calculating their geometric mean before entering the value in the AHP decision-making software to calculate the priorities [84]. For example, in a two people subcommittee using the fundamental scale shown in Table 4.7, if one expert thinks that *Performance* is equally as important as *Scalability* (intensity 1) and the other thinks that *Performance* is extremely more important than *Scalability* (intensity 9) the geometric mean of these judgments is 3 in favor of *Performance*, i.e., *Performance* is moderately more important than *Scalability*.

According to Mu and Pereyra-Rojas [84] the participants typically agree in the goal of the evaluation and the alternatives or candidates to be evaluated, but there might be conflicts regarding the criteria defined to represent the problem. For example one group of the evaluation team might agree with our hierarchical decomposition in Figure 4.2, but another group might choose different criteria to represent the decision-making problem. The basic solution proposed is to work with both hierarchies in parallel and then choose the alternative that provides best overall benefit for both representations of the problem. Even though these hierarchies will have some differences, in a realistic scenario they should be very similar so the effort required for the evaluation is not duplicated.

5.4 Choosing techniques for collecting data

After trimming the list of potential candidates during the Quick Assessment it is time to collect data about those who passed the initial filtering phase. There are several techniques for collecting data that can be executed for each evaluation attribute generated in Section 4.2 (leaf nodes in Figure 4.2). What technique we choose to use basically depends on two factors: the budget for the evaluation and the risk associated to the evaluation attribute. In [16, Sec. 4.2] the techniques are classified in the three categories: literature reviews, vendor appraisals and hands-on techniques.

Literature reviews are one of the cheapest way for collecting data so they may be attractive for evaluators that wish to reduce the evaluation cost as much as possible. In any case the evaluators should be careful to review only good quality and trusty sources. If the evaluation attribute has a lot of associated risk then it is advisable to complement the literature review with a hands-on technique.

At a first glance vendors appraisals might be considered irrelevant in this evaluation because the middleware has to be open source software. I found that many open source middlewares on the market also provide a commercial license, so a vendor does exist. In the case of *pure* open source middleware, they are sometimes owned or supported by big organizations. Gathering data from these vendors or organizations might be useful. In the full hierarchical representation of the decision problem (Figure 4.2) I included this strategic concern as the high level criterion *Open Source Project Quality* which is inspired by the model proposed in [39]. From several other possibilities I chose this particular model because it seems to provide low cost metrics for assess the quality of open source projects. Of course, these metrics can be tailored (add, remove or replace) to fit the needs of the CTA evaluators and vendor appraisals can be used to collect data about some of these criteria.

The hands-on techniques are the best way to collect trusty and high quality data. They are also the most expensive. If the budget allows it, hand-on experiments should be used to verify the claims and data collected by literature reviews. In any case hands-on techniques should be used for collecting data about those evaluation attributes that represent a high risk for the system success.

I found an interesting relationship between the data collection techniques described in [16] and the metrics terminology suggested by the ISO/IEC 9126 standard [47] which was introduced in Section 5.2. Table 5.3 shows this connection.

Table 5.3: Mapping between metrics types and data collection techniques.
Source: Elaborated by the author

Metrics type	**Data collection techniques**
Internal metrics	● Literature reviews: users newsgroups, web pages, outside evaluation reports, user manuals, documentation, marketing brochures, release notes, version histories and vendor references. ● Vendor/owner appraisals: vendor/owner business and capability information. ● Hand-on techniques: product probes, benchmarking and demonstrations.
External metrics	● Hands-on techniques: test beds, scenario-based evaluations and prototypes.
Quality in use metrics	● Hands-on techniques: product insertion.

In this phase of the ACTL system development is very unlikely that data about quality in use metrics will be collected. Product insertions, which can have a relatively low cost, require that the OTS component can be integrated easily with the other system components [16]. This is definitely not the case in this evaluation because changing the communication middleware will have repercussions in virtually all the components of the system.

As I proposed in Section 5.2, collecting data about external metrics using test beds and prototypes would be the best way to reduce uncertainty and risk because these techniques provide an environment that is close to the real system context, but they are relatively expensive. The recommendation is to use these techniques for collecting data about, at least, the most important criteria that was prioritized in the second PECA step. Fortunately CTA is already using test with prototype systems for decision making about technology choices [19] and they also have test beds such as the one in INAF IASF Bologna server room which is used for testing software component interactions of the ASTRI SST-2M prototype [80].

Data collection techniques for measuring internal quality can be used more freely due to their relative lower cost. Nevertheless collecting massive amounts of data without purpose is not going to help either. I only included *vendor business and capability information* as a source of vendor appraisal source in Table 5.3 because the importance of the vendor in commercial off-the-shelf software is diluted when we are evaluating open source software due to the existence of the community. Instead of gathering high amounts of data about the vendor or owner, the evaluators are encouraged to collect data about the open source community which might be done by examining the open source project statistics and joining the corresponding mailing lists. Wasserman, Pal and Chan [39] proposed several metrics to easily measure the quality of the open source project using mostly literature reviews.

In Section 4.3.2 and Section 4.3.3 I show an example of the use of literature reviews and a hands-on technique respectively for gathering data about internal metrics.

5.5 Avoiding common mistakes in middleware benchmarking

One common mistake that evaluators often commit is to benchmark performance instead of making a real evaluation process, because the former is easier to do. As Henning [48, p. 3] eloquently points out about middleware benchmarking: "[by] having gone through the 'evaluation process' participants feel that they have exercised due diligence, even though the results may well be irrelevant." Moreover, "Synthetic benchmarks can provide only an overall rough indication of the performance of a real application" [48, pp. 30-31]. Vinoski also agrees that focusing blindly only on performance during middleware evaluation is often a mistake: "it's a lot like going out to purchase a new family sports utility vehicle and coming home with a Porsche 911 Turbo: it doesn't have room to actually seat the family, nor is it capable of carrying any cargo or going off-road, but it is the fastest vehicle you can find" [85, p. 89]. I chose to benchmark performance and specifically latency because of three reasons: The communication middleware latency has a direct impact in the control system overhead, performance is important for a system that needs to scale and, as explained in Section 5.4, it is a feasible hands-on technique for data collection about an internal metric in this iteration of PECA. (i.e., we don't need a partial implementation of the system to use this technique).

To avoid making a generic and irrelevant benchmarking I tailored it to better represent the target system using the information gathered from the CTA documentation and the meeting with members of the ACTL team. With the details of the current implementations and prototypes I was able to design a more significative benchmarking as described in the test setup of the benchmarking (see Section 4.3.3).

The most usual mistake during performance benchmarking that I found in the literature is to start collecting the timing data from the beginning of the execution of the experiment [78, 86]. This generates unstable results because the timing of the first operations are often distorted by external factors such as application startup, priming of caches, stabilization of adaptive resource allocation algorithms in response to system load or network traffic and just in time compilation [78, Sec. 2.2]. To analyze the consequences of these issues I measured latency during warm-up time (first 10000 iterations) independently from the latency during runtime (when the speed is stable) and present both results.

Another mistake that practitioners make is to collect a low amount of measures. Benchmarking middleware may be difficult because unexpected and unavoidable factors can distort the results. One clear example is the Java garbage collector mechanism which is fundamental for the correct execution of Java-based middleware, but introduces non-deterministic distortions to the timing data [86]. To mitigate the effect of these kind of factors, data about 100000 interactions was collected. Even though the benchmark was executed in a closed network I couldn't isolate the experiment from external software working on the same network used for the benchmarking. To mitigate this effect I ran the benchmark in a period with very low activity (stable network) and executed the whole experiment five times.

Some middleware technologies might not perform some actions when the user issues them. This may happen when we unmarshal complex data types that provide overloadable access operators or methods [78, Sec. 2.5]. In this benchmarking a structure composed by simple data types was used as the parameter of the message.

The final mistake I address in this section is the use of software tools for measuring time in a wrong resolution. "Benchmarking CORBA middleware typically involves measuring actions that are as short as tens of microseconds" so the sources with millisecond or worse resolution are inadequate [78, Sec. 2.1]. By doing a little research I found that Python 3 provides `time.perf_counter` as a performance counter with the highest available resolution to measure a short duration. On the other hand Java provides `System.nanoTime`. The time was only measured in the client processes so a C++ time library wasn't needed.

5.6 Guidelines for choosing the best AHP approach to consolidate data

The analytic hierarchy process provides two basic approaches to consolidate data [36]: Absolute measurement and relative measurement. The choice between the two approaches is mainly based on the experience and knowledge of the evaluators. Absolute measurement allows the practitioners to evaluate an arbitrarily big list of candidates and adding new candidates does not generate rank reversals, but creating scales, specially for intangible criteria for which no standard scale exist, requires prior knowledge and experience [36, 48].

On the other hand, relative measurement only supports a reduced amount of alternatives because adding more candidates significatively increases the amount of pairwise comparisons required. Furthermore adding new alternatives to the evaluation might generate rank reversals. Nonetheless, this approach is feasible for more unexperienced evaluators. Moreover, Saaty stated that "for most evaluations with no precedent, the relative measurement is the only meaningful approach" [36, Sec. 1.4].

CHAPTER 6: CONCLUSIONS

Due to the explosive growth of open-source software, organizations are relying more and more on this COTS-like software to assist their development of complex systems. A straightforward advantage of using these technologies is the reduction of the short term cost of building a software system. Even though the OTS software provides some functionality that otherwise must be coded by the development team, their use requires at least two additional tasks: selection and integration. The evaluation and selection of the appropriate open-source products is often an underestimated and overlooked task. The developers are happy because they have several alternatives to use and they will avoid great chunk of coding, but they forget that coding is replaced by selection and integration. If we don't do a proper job at selecting the OTS products then we are just transforming short term cost into long-term cost (i.e., the integration effort required will increase and the developers will have to fill several gaps of the OTS software to fulfill the requirement specification of the system). Furthermore, it may be the case that the long-term cost induced by the inadequate selection of OTS software is so high that the organization is unable to pay it and the project ultimately fails.

Even though choosing the most trending or the *coolest* software of the moment might be enough for the selection of trivial software, this is definitely not the case when we are choosing a communication middleware. Some middleware products can be considered as complex distributed systems on their own right and it is naive to think that we can arbitrarily select one and just make it work because the cost of integrating such an OTS software and the risk of the system failing are too high. We must follow an engineering approach to select communication middleware which is especially beneficial for long-term projects such as ACS and CTA.

In this work I propose a solution to the problem of communication middleware selection. It is based on PECA which is a process for evaluation and selection of COTS software. This methodical and systematic approach allow the evaluators to tailor the process according to their needs and budget, so they have enough control to decide how much resources are willing to spend in the short term to reduce the costs and the risk in the long term. In Chapter 4 I validated this solution by applying it to the ACS's communication middleware selection for CTA. A common deficiency of the selection processes in the literature is that they often don't explain in detail to the evaluators what they have to do and how to do it. In this dissertation I mitigated this deficiency by executing a first iteration of PECA that might serve as a guide for the full execution of the process.

Managing trade-offs is an integral part of software development of distributed systems. These trade-offs occur at different levels in the projects and should be identified and highlighted as soon as possible because they are often a source of conflicts between stakeholders and a source of system failures when they are not addressed correctly. Unfortunately, as Henning stated, "Modern middleware has reached a level of complexity where only a handful of experts have the experience to correctly judge the relative merits of various design and feature trade-offs" [48, p. 3].

Even though I am not an expert in distributed system, I was able to identify several important trade-offs just by executing the evaluation and selection process proposed in this dissertation. For example, CORBA is arguably slowing down ACS as its communication middleware [10]. In my latency benchmarking I noted that the current implementations of CORBA are not that slow when we compare them to some of their competitors (see Section 4.3.3). So, What is happening? ACS does not only use CORBA, but it also implements a custom Component Container Model (CCM) [7]. This CCM attempts to separate the functional from the technical concerns and facilitating the deployment and administration by increasing the system transparency (i.e., hiding low level details of technologies used underneath such as CORBA). It is known that in distributed systems there is a trade-off between transparency and performance. Maybe using the CCM was a good idea in 2004, but today it should be reevaluated.

The CCM was created for a number of reasons such as facilitating the software development of a complex system by domain experts that are not necessarily technical experts by masking the complexity of CORBA. Although the origin of this complexity can be argued about, one of the main reasons to use CORBA in the first place is because it provides high interoperability in terms of programing languages, software and hardware platforms supported. Modern observatories don't need such degree of interoperability. Homogeneous hardware and software can be acquired more easily now. CTA even introduced an interoperability layer by using OPC UA middleware to homogenize the access to hardware devices. So this apparent trade-off between interoperability and complexity that CORBA impose is not so convenient anymore. If we can get the right amount of interoperability (language interoperability is still important) and the right amount of complexity by replacing CORBA for another technology, then we might analyze the alternatives to the use of a CCM that are able to balance better the low-complexity needs with the high-performance requirements.

On the other hand, performance is just a variable in the whole equation. What about the high availability and reliability requirements of CTA? There is also a trade-off between performance and reliability. ZeroMQ was the slowest technology in my latency benchmarking, but it is known that it can handle network problems and reconnect the client and server when the problems are solved. Even more, there is a trade-off between the system performance and the level of abstraction in which developers works. Ice allows the developers to work at a high level of abstraction with reliable implementations of standard services which may be difficult (or expensive) to implement from scratch using low level protocols like TCP and ZeroMQ in a reliable way. Nonetheless, sometimes we also need to work at a low level of abstraction to get the required performance (e.g., for CTA bulk data transfer). As was stated during the coupling analysis in Section 4.4.3, centralized administrative information reduce the coupling of the system making it more extensible, but at the same time, centralized data affects negatively to the scalability of the system [24].

Distributed control systems are inherently complex. Fortunately, besides the selection of an appropriate middleware, the execution of PECA also allows the evaluators to improve their insight about the system making this process the perfect opportunity for senior engineers and experts to share their knowledge with the rest of the organization and even training junior engineers that might be included in the evaluation team. Moreover, by properly documenting the process, the evaluation team can also share its improved know-how about making evaluations (e.g., useful evaluation criteria and data collecting techniques). These documents form part of the reuse infrastructure of the organization and they can be used to support a new iteration of PECA or even the evaluation of another relevant OTS software products.

PECA proved to be a good approach for middleware evaluation and selection, but it has some limitations. For instance it relies on human experience to tailor de process and requires expert support for handling the mismatches of the OTS product. By being a tailorable process PECA also inherits the limitations of the techniques used during its execution. The use of pairwise comparison as described by AHP to prioritize the criteria and generating scores of the candidates also relies on the judgment of knowledgeable evaluators. Even though these limitations may not affect CTA, they might be showstoppers for other organizations that lack the necessary personnel and resources for conducting this process.

These shortcomings are a source of future research. For example, AHP pairwise comparisons might be replaced by another technique that does not relies so heavily in expert availability. Moreover, AHP is a well-known approach for solving MCDM problems that has being researched for several years. In this dissertation I used a classic implementation of AHP, yet many variants of the process has being proposed over the years and their application to this particular problem can be analyzed. On the other hand, due to the partial execution of PECA, there is still a lot of work to be done. A research about good metrics for the evaluation attributes presented in Figure 4.2 and their respective measurement methods would be an excellent complement to this effort.

The development of complex distributed systems is often a roadmap filled with obstacles that we must overcome. This dissertation allow us to make a step on this direction by supporting engineers in the execution of a fundamental task for assuring the success of the systems based on the reuse of OTS software.

REFERENCES

[1] J. Ibsen *et al.*, “Building a world-wide open source community around a software framework: progress, dos, and don’ts,” in *Software and Cyberinfrastructure for Astronomy IV*, 2016, vol. 9913, p. 99131D.

[2] M. Araya, L. Pizarro, and H. von Brand, “Packaging and High Availability for Distributed Control Systems,” in *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017*, 2018, p. THPHA045.

[3] C. E. Menay *et al.*, “New architectures support for ALMA common software: lessons learned,” in *Software and Cyberinfrastructure for Astronomy*, 2010, vol. 7740, p. 77401S.

[4] W. Hofmann, Ed., “ACTL requirements,” CTA Consortium, MAN-PO/121201, May 2013.

[5] M. Araya *et al.*, “A New ACS Bulk Data Transfer Service for CTA,” in *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017*, 2018, p. THBPL03.

[6] M. Hiramatsu, “Special Contribution ALMA and its Observational Achievements: Unveiling the Mysteries of the Dark Universe,” *Fujitsu Sci. Tech. J.*, vol. 53, no. 3, pp. 9–14, 2017.

[7] G. Chiozzi *et al.*, “The ALMA common software: a developer-friendly CORBA-based framework,” in *Advanced Software, Control, and Communication Systems for Astronomy*, 2004, vol. 5496, pp. 205–219.

[8] G. Chiozzi, H. Sommer, and J. Schwarz, “ALMA common software architecture,” *ALMA Project document COMP70*, vol. 25, 2009.

[9] M. Henning, “The Rise and Fall of CORBA,” *Queueing Syst.*, vol. 4, no. 5, pp. 28–34, Jun. 2006.

[10] A. Dworak, M. Sobczak, F. Ehm, W. Sliwinski, and P. Charrue, “Middleware trends and market leaders 2011,” in *Conf. Proc.*, 2011, vol. 111010, p. FRBHMULT05.

[11] J. Avarias, H. Sommer, and G. Chiozzi, “Data Distribution Service as an alternative to CORBA Notify Service for the ALMA Common Software,” in *Proceedings of 12th International Conference on Accelerator and Large Physics Control Systems (ICALEPCS)], Advanced Telescope and Instrumentation Control Software II*, 2009.

[12] J. A. Avarias, J. S. López, C. Maureira, H. Sommer, and G. Chiozzi, “Introducing high performance distributed logging service for ACS,” in *Software and Cyberinfrastructure for Astronomy*, 2010, vol. 7740, p. 77403G.

[13] B. Jeram, G. Chiozzi, R. J. Tobar, M. Watanabe, and R. Amestica, “Reimplementing the bulk data system with DDS in ALMA ACS,” *ICALEPCS, TUCOCB08, San Francisco, USA*, 2013.

[14] H. Dai, S. Mishra, and M. A. Hiltunen, “CORBA-as-Needed: A Technique to Construct High Performance CORBA Applications,” in *High Performance Computing — HiPC 2002*, 2002, pp. 141–150.

[15] ACTL Team and Others, “Array Control & Data Acquisition Technical Design Report,” CTA Consortium, Tech. Rep. ACTL-TDR/140415, May 2015.

[16] S. Comella-Dorda, J. Dean, G. Lewis, E. Morris, P. Oberndorf, and E. Harper, “A Process for COTS Software Product Evaluation,” CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, CMU/SEI-2003-TR-017, Jul. 2004.

[17] A. Orlati *et al.*, “Evolution in the Development of the Italian Single-dish COntrol System (DISCOS),” in *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017*, 2018, p. THPHA014.

[18] T. C. Consortium, “Cherenkov Telescope Array: The Next Generation Gamma-ray Observatory,” *arXiv preprint arXiv:1709.05434*, 2017.

[19] M. Füßling *et al.*, “Status of the array control and data acquisition system for the Cherenkov Telescope Array,” in *Software and Cyberinfrastructure for Astronomy IV*, 2016, vol. 9913, p. 99133C.

[20] V. Sliusar *et al.*, “Control Software for the SST-1M Small-Size Telescope prototype for the Cherenkov Telescope Array,” *arXiv [astro-ph.IM]*, 13-Sep-2017.

[21] E. Antolini and Others, “Telescope Control System of the ASTRI SST-2M prototype for the Cherenkov Telescope Array,” in *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017*, 2018, p. THMPL04.

[22] M. Völter, M. Kircher, and U. Zdun, “Introduction To Distributed Systems,” in *Remoting patterns: foundations of enterprise, internet and realtime distributed object middleware*, John Wiley & Sons, 2013, pp. 1–18.

[23] Q. Mahmoud, “Introduction,” in *Middleware for Communications*, John Wiley & Sons, 2005, pp. 30–37.

[24] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[25] M. F. Alrahmawy, “Real-Time Middleware,” *no. September*, 2010.

[26] D. C. Schmidt, “Real-time CORBA with TAO (The ACE ORB),” *Washington University in St. Louis - Computer Science & Engineering at WashU*, 13-Jan-2013. [Online]. Available: http://www.cs.wustl.edu/~schmidt/TAO.html. [Accessed: 20-May-2018].

[27] “JacORB,” *JacORB*, 31-Aug-2017. [Online]. Available: https://www.jacorb.org/. [Accessed: 19-May-2018].

[28] D. Grisby, “Free CORBA ORB,” *omniORB*, 11-May-2018. [Online]. Available: http://omniorb.sourceforge.net/. [Accessed: 19-May-2017].

[29] X. Liu, C. Combe, and B. Wang, “Component-based development.” Google Patents, 2008.

[30] R. Garg, “A systematic review of COTS evaluation and selection approaches,” *Accounting Finance*, vol. 3, no. 4, pp. 227–236, 2017.

[31] V. Bali and S. Madan, “COTS Evaluation & Selection Process in Design of Component based Software System: An Overview and Future Direction,” *Global Journal of Computer Science and*, 2015.

[32] F. Tarawneh, F. Baharom, J. H. Yahaya, and F. Ahmad, “Evaluation and selection COTS software process: The state of the art,” *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 1, no. 2, pp. 344–357, 2011.

[33] K. Basir, A. Khanum, F. Azam, and A. Qavi, “TAES-COTS: Thorough Approach for Evaluation & Selection of COTS Products,” in *Frontiers of Information Technology (FIT), 2014 12th International Conference on*, 2014, pp. 91–96.

[34] A. Mohamed, G. Ruhe, and A. Eberlein, “COTS selection: past, present, and future,” in *Engineering of Computer-Based Systems, 2007. ECBS’07. 14th Annual IEEE International Conference and Workshops on the*, 2007, pp. 103–114.

[35] J. Kontio, G. Caldiera, and V. R. Basili, “Defining Factors, Goals and Criteria for Reusable Component Evaluation,” in *Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative Research*, 1996, p. 21.

[36] T. L. Saaty, “Fundamentals of decision making and priority theory with the analytic hierarchy process (Analytic Hierarchy Process Series, Vol. 6),” *Auflage, Pittsburg*, 2000.

[37] M. Brunelli, “Introduction and fundamentals,” in *Introduction to the Analytic Hierarchy Process*, Springer, 2014, pp. 7–20.

[38] T. L. Saaty, “How to Make a Decision: The Analytic Hierarchy Process,” *Interfaces* , vol. 24, no. 6, pp. 19–43, Dec. 1994.

[39] A. Wasserman, M. Pal, and C. Chan, “The business readiness rating model: an evaluation framework for open source,” in *Proceedings of the EFOSS Workshop, Como, Italy*, 2006.

[40] X. Burgués, C. Estay, X. Franch, J. A. Pastor, and C. Quer, “Combined Selection of COTS Components,” in *COTS-Based Software Systems*, 2002, pp. 54–64.

[41] E. H. Forman, “Facts and fictions about the analytic hierarchy process,” *Math. Comput. Model.*, vol. 17, no. 4, pp. 19–26, Feb. 1993.

[42] R. Whitaker, “Criticisms of the Analytic Hierarchy Process: Why they often make no sense,” *Math. Comput. Model.*, vol. 46, no. 7, pp. 948–961, Oct. 2007.

[43] A. Emrouznejad and M. Marra, “The state of the art development of AHP (1979--2017): a literature review with a social network analysis,” *Int. J. Prod. Res.*, vol. 55, no. 22, pp. 6653–6675, 2017.

[44] G. Raffi, B. Glendenning, J. Schwarz, B. Glendenning, R. Sramek, and C. Haupt, “ALMA Common Software Technical Requirements,” *Organization*, vol. 2005, pp. 09–26, 2005.

[45] A. Gokhale and D. C. Schmidt, “Measuring the Performance of Communication Middleware on High-speed Networks,” in *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Palo Alto, California, USA, 1996, pp. 306–317.

[46] W. B. Goethert, E. K. Bailey, and M. B. Busby, “Software Effort and Schedule Measurement: A framework for counting Staff-hours and reporting Schedule Information,” CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, CMU/SEI-92-TR-021, Sep. 1992.

[47] *Software engineering -- Product quality -- Part 1: Quality model*, ISO/IEC 9126-1:2001, Jun. 2001.

[48] M. Henning, “Choosing middleware: Why performance and scalability do (and do not) matter,” *Whitepaper*, 2009.

[49] *IEEE Standard Glossary of Software Engineering Terminology*, 610.12-1990, Dec. 1990.

[50] I. Wijegunaratne and G. Fernandez, *Distributed Applications Engineering: Building New Applications and Managing Legacy Applications with Distributed Technologies*. Springer Science & Business Media, 2012.

[51] T. L. Saaty and L. G. Vargas, *Models, Methods, Concepts & Applications of the Analytic Hierarchy Process*. Springer Science & Business Media, 2012.

[52] B. K. Jayaswal, P. C. Patton, and E. H. Forman, *The Analytic Hierarchy Process (AHP) in Software Development (Digital Short Cut)*. Pearson Education, 2007.

[53] G. Hohpe and B. Woolf, “Integration Styles,” in *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2004, pp. 39–56.

[54] D. G. M. Garcia, “Event Management and Composition Framework for Building Adaptive Systems,” 2006.

[55] Inspirel, “YAMI4 vs. ZeroMQ,” *Inspirel*, Apr-2017. [Online]. Available: http://www.inspirel.com/articles/YAMI4_vs_ZeroMQ.html. [Accessed: 24-Dec-2017].

[56] P. Wegner *et al.*, “The software system for the Control and Data Acquisition for the Cherenkov Telescope Array,” in *Journal of Physics: Conference Series*, 2016, vol. 762, p. 012030.

[57] A. Dworak, F. Ehm, P. Charrue, and W. Sliwinski, “The new CERN controls middleware,” in *Journal of Physics: Conference Series*, 2012, vol. 396, p. 012017.

[58] TANGO Consortium, “What is Tango Controls ?,” *What is Tango Controls ? - TANGO Controls*, n.d. [Online]. Available: http://www.tango-controls.org/what-is-tango-controls/. [Accessed: 23-May-2018].

[59] A. M. Riad, A. E. Hassan, and Q. F. Hassan, “Investigating performance of XML Web services in real-time business systems,” *Journal of Computer Science and System Biology*, vol. 2, pp. 266–271, 2009.

[60] The Open MPI Project, “FAQ: General information about the Open MPI Project,” *Open MPI: Open Source High Performance Computing*, 08-May-2018. [Online]. Available: https://www.open-mpi.org/faq/?category=general. [Accessed: 19-May-2018].

[61] Real-Time Innovations, “Develop and Integrate Systems with Connext DDS Professional | RTI Products,” *RTI*, n.d. [Online]. Available: https://www.rti.com/products/dds. [Accessed: 19-May-2018].

[62] Object Computing, Inc., “OpenDDS,” *OpenDDS*, n.d. [Online]. Available: http://opendds.org/. [Accessed: 19-May-2018].

[63] P. Hintjens, “ØMQ - The Guide,” *Distributed Messaging - zeromq*, n.d. [Online]. Available: http://zguide.zeromq.org/page:all. [Accessed: 13-Jan-2018].

[64] Inspirel, “YAMI4 - Messaging Solution for Distributed Systems,” *Inspirel*, n.d. [Online]. Available: http://www.inspirel.com/yami4/. [Accessed: 19-May-2018].

[65] IBM, “IBM Knowledge Center,” *IBM - United States*, 19-May-2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.pro.doc/q001020_.htm. [Accessed: 19-May-2018].

[66] Apache Software Foundation, “Apache Thrift - Home,” *Apache Thrift - Home*, n.d. [Online]. Available: https://thrift.apache.org/. [Accessed: 19-May-2018].

[67] Google Inc., “gRPC open-source universal RPC framework,” *GRPC*, n.d. [Online]. Available: https://grpc.io/about/. [Accessed: 19-May-2018].

[68] EPICS V4 Working Group, “EPICS Version 4 Home Page,” *EPICS Version 4 Home Page*, n.d. [Online]. Available: http://epics-pvdata.sourceforge.net/. [Accessed: 19-May-2018].

[69] Oracle Corporation, “An Overview of RMI Applications,” *Lesson: All About Sockets (The Java^{TM} Tutorials > Custom Networking)*, n.d. [Online]. Available: https://docs.oracle.com/javase/tutorial/rmi/overview.html. [Accessed: 19-May-2018].

[70] ZeroC ,Inc., “Ice,” *ZeroC, Inc*, n.d. [Online]. Available: https://zeroc.com/products/ice. [Accessed: 19-May-2018].

[71] .NET Foundation, “What Is Windows Communication Foundation,” *Microsoft Docs*, 30-Mar-2017. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf. [Accessed: 20-May-2018].

[72] The Apache Software Foundation, “Apache CXF^{TM}: An Open-Source Services Framework,” *Apache CXF*, 16-Apr-2018. [Online]. Available: http://cxf.apache.org/. [Accessed: 21-May-2018].

[73] Google Inc., “Protocol Buffers | Google Developers,” *Google*, n.d. [Online]. Available: https://developers.google.com/protocol-buffers/. [Accessed: 21-May-2018].

[74] *Systems and software engineering -- Vocabulary*, ISO/IEC/IEEE 24765:2010, Dec. 2010.

[75] M. Zenger, "PROGRAMMING LANGUAGE ABSTRACTIONS FOR EXTENSIBLE SOFTWARE COMPONENTS," ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2004.

[76] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979.

[77] M. Henning and M. Spruiell, "Distributed programming with ice," *ZeroC Inc. Revision*, vol. 3, p. 97, 2003.

[78] A. Buble, L. Bulej, and P. Tuma, "Corba benchmarking: A course with hidden obstacles," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 6–pp.

[79] A. Krüger and B. Neppert, "Software Architecture for ACTL," CTA Consortium, Aug. 2015.

[80] V. Conforti and Others, "Procedures of Software Integration Test and Release for ASTRI SST-2m Prototype Proposed for the Cherenkov Telescope Array," in *Proceedings, 16th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS 2017): Barcelona, Spain, October 8-13, 2017*, 2018, p. TUPHA004.

[81] I. Sadeh, I. Oya, J. Schwarz, E. Pietriga, and D. Dezman, "The Graphical User Interface of the Operator of the Cherenkov Telescope Array," in *Proceedings of the 2017 ICALEPCS conference*, 2017.

[82] I. Millet and T. L. Saaty, "On the relativity of relative measures--accommodating both rank preservation and rank reversals in the AHP," *Eur. J. Oper. Res.*, vol. 121, no. 1, pp. 205–212, 2000.

[83] W. Ossadnik and R. Kaspar, "Evaluation of AHP software from a management accounting perspective," *Journal of Modelling in Management*, vol. 8, no. 3, pp. 305–319, 2013.

[84] E. Mu and M. Pereyra-Rojas, *Practical Decision Making using Super Decisions v3: An Introduction to the Analytic Hierarchy Process*. Springer, 2017.

[85] S. Vinoski, "The performance presumption [middleware evaluation]," *IEEE Internet Comput.*, 2003.

[86] C. Demarey, G. Harbonnier, R. Rouvoy, and P. Merle, "7 BENCHMARKING THE ROUND-TRIP LATENCY OF VARIOUS JAVA-BASED MIDDLEWARE PLATFORMS," 2015.

APPENDIX A (PECA STEPS)

1. **Planning the evaluation**
    a. Forming the evaluation team
        i. Include people with diverse skills (e.g., technical experts, domain experts, business analyst and end users)
    b. Creating the charter
        i. Evaluation goal and scope
        ii. Team members description
        iii. State of commitment of evaluators and decision makers
        iv. Factors that limit the selection
        v. Summary of decisions already being made
    c. Identifying Stakeholders
        i. Avoid errors of inclusion or exclusion
        ii. Identify their expectations
    d. Picking the approach
        i. Assess the complexity of the evaluation and risk of failure and determine if they justify a methodical evaluation process (such as PECA)
        ii. Choose between *first fit* or *best fit* approach
        iii. Define some initial filters if the number of potential candidates is too high
    e. Estimating Resources and Schedule
        i. Consider factors like level of rigor, risk involve, number of candidates and the evaluators level of experience

2. **Establishing the criteria**
    a. Defining the evaluation requirements
        i. Some sources are: system requirements, architecture/interface constraints, programmatic constraint, operational environment and stakeholders expectations
        ii. Avoid error of inclusion and exclusion
        iii. Determine if the requirements are mandatory or negotiable
    b. Defining the evaluation criteria
        i. Decompose the evaluation requirement into criteria (capability statements) that the alternatives should comply
        ii. Verify that each criteria is evaluable, significative, non-overlapping and have the ability to discriminate between the candidates
    c. Establishing priorities

    i. Make judgements based on your understanding of the system to assign priorities to the criteria using some technique (e.g., unstructured weighing, Delphi technique or pairwise comparison).

3. **Collecting data**
    a. Literature Reviews
        i. Sources: Internet, reports from outside evaluators (beware of the different system context), user manuals, marketing brochures, release notes, version histories, etc
    b. Vendor Appraisals
        i. Sources: Interviews, vendor literature, capability evaluations, financial analyses (external), strategic information and list of users, customer compliments and complaints, etc
    c. Hands-on Techniques
        i. Techniques: test beds, product probes, prototypes, scenario-based evaluations, benchmarking, product insertion and demonstrations

4. **Analysing the data**
    a. Consolidation of data
        i. Merge the qualitative and quantitative data into useful information (like a score of fitness for each candidate) using some technique
    b. Analyse the consolidated information
        i. Reason about the information generated previously. Use some techniques to help you in this task
    c. Making the recommendations
        i. Predict what information the decision makers need and provide it to them
        ii. Document the data gathered and the knowledge generated during the evaluation. You might adapt the templates in Appendix B, Appendix C and Appendix D to your particular needs.

APPENDIX B (PRODUCT DOSSIER TEMPLATE)

1. **Product Identification**
    a. Name
    b. Version number, rev number, patches installed, etc.
2. **Vendor Contact Information**
3. **Product Description**

[Summary of what the product does and what it is being considered for/how it is used in the system.]

4. **Product Status**

[Current state of decisions made regarding use of the product, whether it has been selected, is being used, actively maintained, or being replaced/retired.]

5. **State of Evaluation, Testing, Certification**
6. **Vendor Data (includes raw and processed information)**
    a. Financial
    b. Business
    c. Engineering
7. **Product Data (includes raw and processed information)**
    a. Basic characteristics
    b. Standards
    c. Hardware/software configuration required
    d. Functional capabilities
    e. Nonfunctional capabilities [usability, supportability, interoperability, reliability, security, etc.]
    f. Interactions and behavior
    g. Performance
    h. Documentation
    i. Licensing
    j. Architecture
    k. Noted discrepancies between the product and its documentation
8. **Product Limitations**
    a. Product deficiencies
    b. Limitations on product use
9. **System Relationships, Tailoring, and Modifications (includes raw and processed information)**
    a. System configuration
    b. System adaptation
    c. System integration
    d. Product and system tailoring and modification
    e. Design strategies for using product

10. **Product Usage History**
    a. Dates considered, used, retired
    b. Bugs/problems reported
    c. Disposition of bugs/problems
    d. Queries to vendor or third parties for support
    e. Changes/updates to configurations and tailoring [capture rationale, changes, and results]
    f. Preventative/other maintenance performed
11. **Dossier Usage History**
    a. Who, what, and why record of access to Dossier components
    b. Errata or inconsistencies found [additional information required]

APPENDIX C (EVALUATION RECORD TEMPLATE)

1. **Charter**
    - 1.1. Background
        - 1.1.1. a. Date of effort
        - 1.1.2. b. Evaluation team members and qualifications
        - 1.1.3. c. Facilities and resources used
    - 1.2. System Stakeholders

Stakeholder	Requirements	Sponsorship, administration	Contractual information	Technical information	Other

- 1.3. Approach
    - 1.3.1. Depth
        - 1.3.1.1. Complexity
        - 1.3.1.2. Risk of failure
    - 1.3.2. First fit vs. best fit
    - 1.3.3. Number and type of filters
    - 1.3.4. Other

2. **Criteria Record**

Criterion	Negotiability Non applicable Very negotiable Negotiable Barely Negotiable Hard Requirement	Capability Statement	Measurement Method

3. Results Record

Criterion	<product1 name>		<product2 name>	
	Measurement Results	Repair Strategy(s) and Cost of Fulfillment	Measurement Results	Repair Strategy(s) and Cost of Fulfillment

4. Assessment of Evaluation Effort

4.1. Limitations or deficiencies

4.2. Rationale for all decisions made

APPENDIX D (SUMMARY OF RESULTS TEMPLATE)

1. **Analysis of fitness**
    a. Observed product features and limitations
    b. Performance against criteria for all candidates
    c. Behaviour and interactions with other components
    d. Results of gap analysis, fulfillment strategies and fulfillment costs
    e. Results of sensitivity analysis
2. **Analysis of evaluation deficiencies**
    a. Need for further evaluation
    b. Confidence in results
3. **Discovered facts**

[Differences between product and documentation, for example]

4. **Others (Optional)**
    a. recommended products or alternate solutions, with rationale and system implications
    b. if the recommendation is to *use*, recommendations regarding
        i. architecture, design, implementation
        ii. tailoring or wrapping
        iii. integration and system testing
        iv. maintenance and support
    c. if the recommendation is not to *use*
        i. alternatives (such as build, change requirements, etc.)