A Pattern Language for Designing Application-Level Communication Protocols and the Improvement of Computer Science Education through Cloud Computing

Jorge Edison Lascano
Utah State University

A PATTERN LANGUAGE FOR DESIGNING APPLICATION-LEVEL COMMUNICATION PROTOCOLS AND THE IMPROVEMENT OF COMPUTER SCIENCE EDUCATION THROUGH CLOUD COMPUTING

by

Jorge Edison Lascano

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

UTAH STATE UNIVERSITY
Logan, Utah

2017

ABSTRACT

A Pattern Language for Designing Application-Level Communication Protocols and the Improvement of Computer Science Education through Cloud Computing

by

Jorge Edison Lascano, Doctor of Philosophy

Utah State University, 2017

Major Professor: Stephen W. Clyde, Ph.D.
Department: Computer Science

Networking protocols have been developed throughout time following layered architectures such as the Open Systems Interconnection model and the Internet model. These protocols are grouped in the Internet protocol suite. Most developers do not deal with low-level protocols, instead they design application-level protocols on top of the low-level protocol. Although each application-level protocol is different, there is commonality among them and developers can apply lessons learned from one protocol to the design of new ones. Design patterns can help by gathering and sharing proven and reusable solution to common, reoccurring design problems. The Application-level Communication Protocols Design Patterns language captures this knowledge about application-level protocol design, so developers can create better, more fitting protocols base on these common and well proven solutions.

Another aspect of contemporary development technics is the need of distribution of software artifacts. Most of the development companies have started using Cloud

Computing services to overcome this need; either public or private clouds are widely used. Future developers need to manage this technology infrastructure, software, and platform as services.

These two aspects, communication protocols design and cloud computing represent an opportunity to contribute to the software development community and to the software engineering education curriculum. The Application-level Communication Protocols Design Patterns language aims to help solve communication software design. The use of cloud computing in programming assignments targets on a positive influence on improving the Analysis to Reuse skills of students of computer science careers.

(181 pages)

PUBLIC ABSTRACT

A Pattern Language for Designing Application-Level Communication Protocols and the Improvement of Computer Science Education through Cloud Computing

Jorge Edison Lascano

Enterprises that develop software use current technology because of its proven advantages and to accelerate and improve the software development process. Nevertheless, it is difficult to be up-to-date for most professionals in the area. Although students from higher academic institutions need to learn these new tools, and their main purpose is to learn how learn; colleges still need to prepare students for modern enterprise requirements, so they teach new technologies to improve students' skills. Ubiquitous computing is software and services available everywhere, for example in mobile devices, in different locations, in different networks. This computing requires good communication protocols so the software systems can interconnect properly among them to share data and for distribution purposes.

Developers design communication protocols in repetitive times and learn how to deal with different non-functional communication requirements such as reliability, security or synchronicity. This knowledge is typically collected and presented in design pattern languages, they allow professionals to communicate ideas, problems and solutions for communication software implementation. The application-level Communication Design Pattern language aims to gather this knowledge and make it available to computer science novices and experts.

Most modern software applications need to be available in the network.

Developers use cloud computing resources such as virtual infrastructure, software and platforms. These services are available anywhere (anyware). Cloud computing is gradually becoming part of current Computer Science curriculums. Ultimately, we show that using cloud computing resources in programming assignments helps students improve their analysis to reuse skills.

ACKNOWLEDGMENTS

I would like to thank my advisor, Stephen Clyde, for his invaluable support. His academic and professional knowledge has been greatly appreciated and noted from the start to the end of my journey as a researcher at Utah State University. He has also been super supportive in non-academic issues; he is a great person.

I would especially like to thank my committee members for their support and assistance throughout the entire process. I give special thanks to the CS staff Vicki, Cora, Genie, who helped on the data gathering for my research and every other requirement.

I would like to thank my Logan and USU friends and my entire family for their encouragement, moral support, and patience as I worked my way to the graduation. I could not have done it without all of you, especially without the help of my brave wife Catherine, my lovely mother Isabel, my dear brother René, my encouraging sister-in-law Grace, my loved nephew Matias, and my life-inspiring and reasons to go forward every day -my children, Melody, Daniel, and Jorge.

Last but not the least, to USA and Ecuador's people, to every one of my students and friends who supported me in many ways.

Jorge Edison Lascano

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Pattern Languages and Cloud Computing are methods and practices widely used in the different stages of the software development cycle. They have become important for Computer Science professionals in all areas. This represents an opportunity for academic institutions that want to improve their curriculum to prepare the students according to the enterprise needs. The proposed research is a multi-paper dissertation that describes the establishing a pattern language for designing Application-level Communication Protocols (ACPs) for Distributed Systems. It also deals with improving the Computer Science (CS) curricula by including Cloud Computing (CC) in programming assignments.

Since the introduction of software design patterns in the book, “Design Patterns: Elements of Reusable Object Oriented Software” [1], researchers have defined many pattern languages to help developers in addressing common software-engineering problems in areas, such as programming [1], security [2] [3], enterprise applications [4], big data [5], reliable messaging [6], management processes [7], user interfaces [8]. To avoid repeating mistakes and to improve the quality of their software, developers need to understand and use these diverse design patterns. Yet, there is no pattern language currently in the area of ACP design. This is a deficiency that the proposed research hopes to fill. See Chapters 2 and 3.

Another aspect that is becoming important for computer science professionals is Cloud Computing. In general, CC offers remote on-demand computing resources, such as infrastructure, software, and platform services. In industry, most developers are already deploying their applications to some kind of cloud. According to RightScale [9], 95% of 1060 surveyed technical professionals are using some kind of cloud, which implies that the majority of the software being deployed these days involves CC and network communications. Therefore, there is a tremendous need for CS professionals with CC skills. Sections 3 and 4 discuss more details about CC and the possibility of integrating it into CS courses. Part of this proposed research aims to study the impact of such changes in a CS curriculum.

1.1. Design Patterns for Communication Protocols

Most contemporary software applications rely on network communications and, therefore, their developers need to design and implement ACPs. Even though, there are many standard low-level communication protocols available, developers have to design ACPs for every new software system [10]. There are proven practices that could help developers in designing ACPs. That knowledge has not been properly documented and is not easily accessible. This is exactly what design patterns do for developers, prior to this research, there was no pattern language for ACP design [10].

In a paper titled, “A Pattern Language for Application-level Communication Protocols” [10], I presented an initial set of patterns for a new pattern language, called CommDP. This pattern is specifically oriented towards the design of ACPs. The paper was published in ICSEA 2016 conference [11] and received the best paper award. Consequently, I was invited to write a following paper for their journal. Currently, CommDP consists of nine ACPs design patterns: Request-Reply -RR, Request-Reply-

Acknowledge -RRA, Idempotent Retry -IPR, Intermediate State Message -ISM, Second Channel -2Ch, Front End -FE, Proxy -PXY, Reliable Multicast -RMC, and Publish-Subscribe –P-S. They are briefly described in chapter 2, stating the problem they intend to solve, their solution, a sequence of message interchanges, their main characteristics needs (qualities), and their behavior. Nevertheless, to completely define a Pattern Language so it can be reusable, concrete examples and a template, that help unify the information for every pattern are important [1], also we need a guideline to use the appropriate pattern according to our needs. Thus, I propose the following template elements for defining CommDP [10]: Name and overview, intent, description: (problem, context, solution), consequences, known uses, aliases and related work, examples of application, and references.

In general, the desirable characteristics of network communications are security, fault tolerance, maintainability, openness, extensibility, scalability, and dynamic quality of service [12]. In [10], I present four metrics for ACP design patterns that can help developers decide whether a particular pattern meets their needs and helps them achieve the communication qualities that they desire. These metrics are reliability, synchronicity, longevity, and adaptability for scalable distribution. I ranked each design pattern using for each metric on a scale of 1 to 3, with 3 being the highest. For example, the Proxy Pattern has reliability=1, synchronicity=1, longevity=1, and adaptability=3. If a developer were building a system that required good adaptability for scalable distribution, but is not too much concerned about reliability, synchronicity, or longevity, this pattern would be a good choice. This kind of choices are made at professional level, hence its use in an academic environment will help improve students' skills and knowledge in a realistic environment, this was the case during the Fall-2016 semester, where the instructor of CS5200 used the aforesaid pattern language to teach ACP design and their qualities.

Design patterns and pattern languages are not invents, instead they are collections of proven solutions that have been used widely by different software specialists. Hereafter, a pattern language can be established by an assembly of experts/professionals/academicians in related areas. For this purpose, I have created the CommDP[1] wiki page. Using this tool, this assembly will be able to collaborate to complete the definition of existing patterns, also they could propose to add new patterns. In the second paper, about CommDP, see Chapter 2, I define every pattern in its entirety according to a pattern template. Also, I introduce a guideline to use the respective design pattern according to the stakeholders' needs and the main characteristics of the ACPs.

1.1.1. Literature Review on Pattern Languages for Communication Software

In the last two decades, authors have published pattern languages that deal with Communication Subsystems. Rising [13] presents a compendium of communication software design-pattern articles of different authors expressing their experience on building telecommunications and distributed applications. Li and Chou [14] find patterns in a series of CSTA (Computer Supported Telecommunications Applications) services that are useful to recognize those services and therefore to transform them into RESTful web services. After defining these patterns, the CSTA services are converted into RESTful services by using the CRUD operations from the REST architectural style.

[1] http://commdp.serv.usu.edu/

Daigneau [15] proposes a set of recurrent solutions for the development of SOAP and RESTful web services. These patterns are grouped in six categories, one of them deals with Client-Service interaction (communication) design patterns. Fehling et al. describe a set of Cloud Computing Patterns [16] focusing on the use of cloud resources towards designing, building and managing cloud applications. They extract common behavior and components of cloud products dividing the patterns in four groups: Cloud Offering Patterns, Cloud Application Architecture Patterns, Cloud Application Management Patterns; and Composite Cloud Application Patterns.

One of the most complete works in the area of communication software is by D. Schmidt and others [17], they present a classification of Distributed Real-time and Embedded Systems patterns as a result of two decades of working towards the development of ACE (Adaptive Communication Environment) ORB framework [18] [19]. In [20], they introduce four groups of patterns: Concurrency, Event, Initialization, and Synchronization; plus a set of miscellaneous patterns. This Pattern language is refined later in the book series "Pattern-Oriented Software Architecture" (POSA), Vol.1 [21], vol. 2 [22], vol. 3 [23], and vol. 4 [17]. In the later, they propose a single set of 114 patterns connected to other 150 design patterns from different authors. The patterns are classified in 13 different problem areas: From Mud to Structure, Distribution Infrastructure, Event Demultiplexing and Dispatching, Interface Partitioning, Component Partitioning, Application Control, Concurrency, Synchronization, Object Interaction, Adaptation and Extension, Modal Behavior, Resource Management, and Database Access.

At application level, nevertheless, the work is scattered and the articles are not focused on global solutions for Communication Protocols. There is no author that groups or creates a set of patterns that deal specifically with ACPs.

1.2. Cloud Computing in Software Engineering Education

CC is widely used by industry to improve productivity, scalability, extensibility, and accessibility. Furthermore, the number of organizations adopting CC is increasing daily [9]. This trend presents an opportunity and challenge for higher education to a) incorporate CC technology into curricula so students can be better prepared and b) leverage this technology as means of helping students improve other skills and knowledge required of computer science professionals. These other skills, called Analysis-to-Reuse (A2R) skills, include: analysis, design, tool evaluation, testing, deployment, and reuse [24]. Based on current trends, I believe that CS departments need to integrate CC into programming assignments to help student become familiar with this technology before graduation. I also believe that using CC for programming assignments will improve students' A2R skills. However, whether using CC actually accomplishes these hoped-for objectives is an open question. To answer this question, we need to find initial evidence of CC impact on students' skills. This is one of the primary goals of the proposed research.

To answer this question, the proposed research present an experience of using CC in a distributed systems course, see Chapter 3. Following I have conducted a longitudinal observational study over three semesters to analyze if students who are exposed to CC feel that their A2R skills improved more than students not exposed to CC. This study examines whether integrating CC into programming assignments helps the USU CS Department meet its Program Educational Objective 1 (PEO-1), which states “the USU Computer Science program will prepare its graduates to be successful and contributing professionals by being able to apply the principles of computer science and adapt emerging technologies to analyze and solve real world problems” [25].

The proposed study calls for a systematic integration of CC into a limited number of courses and their programming assignments. Measures are gathered using a Qualtrics[2] survey instrument that collect self-assessment perceptions from students about their CC and A2R skills, see Appendix E. The same survey has been distributed three times after Fall-2015, Spring-2016, and Fall-2016 semesters. My goal is to measure the variances n the students’ perceptions of their A2R skills and knowledge.

The research questions for the study are as follows: 1) what are students’ perceptions about how their own skills and knowledge level change over the course of a semester and 2) how does the integration of CC into programming assignments affect those perceptions. With those questions in mind, I propose my hypotheses: *Students who have assignments that require CC have greater perceived increases in their skills and knowledge levels in the A2R areas than students that are not exposed to the used of CC*.

The study spans the Fall-2015, Spring-2016, and Fall-2016 semesters. I believe that this time is enough to measure change in the students’ perceptions. In Spring-2016 and Fall-2016, the instructors integrated CC into one or more assignments in the following courses: CS5700 (Sp16), CS5700 (F16), CS3450 (F16), and CS5600 (F16).

[2] http://qualtrics.usu.edu/

The data from the surveys is analyzed for trends in a) the students' self-assessments, b) perceptions about improvements in their skills and knowledge, and c) differences in those perceptions between two groups of students: those that had CC integrated into their course assignments (group A) and those who did not (group B), see chapter 4.

During a preliminary study conducted in 2015, the instructor of CS6200/CS7930 used Amazon Web Services[3] (AWS) for distributed application programming assignments, some students used AWS, and other students did not use CC services. This experience led to the paper titled, "Using Cloud Services to Improve Software Engineering Education for Distributed Application Development" [10]. Nevertheless, the sample size was too small to formulate solid conclusions, nevertheless this preliminary study was sufficient to provide guidance on how to setup the proposed observational study. Because this is social study that involve human being, I requested approval for this study to the USU Institutional Review Board –IRB. The proposed study was approved on March 18,206 (See appendix C). The analysis of this study is described in Chapter 4. Currently, this paper has been submitted to the Software Engineering Education and Training Conference [26].

1.2.1. Literature Review on Cloud Computing and Software Engineering Education

The use of virtual resources for computer science courses is not a new concept in the academia [27], most of such use has been with public clouds in courses for networking [28], systems administration [27], distributed applications [29], data processing- MapReduce [30]. Gonzalez et al., at Rochester Institute of Technology used

[3] https://aws.amazon.com/

Amazon EC2 in their Principles of Systems Administration course and leveraged career opportunities for their students [27]. Zhu used cloud resources in the Network Programming course at Metropolitan State University of Denver, the students agreed that using CC resources had a positive impact on their learning experience [28], he states that the same effect will be true for other courses, nevertheless, there is no follow up on this hypothesis. Rabkin et al. [30] used cloud computing for MapReduce measurements at University of California, Berkeley.

Additional to the use of public cloud resources, some universities are using their own virtual environments/clouds to cover engineering educational goals [28] [31]. For example, Syracuse University, has developed SEED, a local virtual machine lab [32]; North Carolina State University, a Virtual Computing Lab [33]; Stony Brook University, V-NetLab [34]; Arizona State University, V-lab [35]. Courses that have been taught using these clouds are: Computer and network security principles, computer networks, distributed systems, grid computing, database administration, cyber security. Nevertheless, programming-oriented classes have not taken advantage of this resource as an educational means. Also, a broader study about the use of CC for improving software engineering education is missing.

The studies presented in this dissertation can contributed to the Computer Science body of knowledge and the Computer Science Curricula defined by IEE and ACM [36], they can also serve as steps forward to improve the quality of the CS program by including CC or other technology topics in a systematic way without affecting negatively in the students learning experience.

This dissertation is a combination of four papers. Two of these have already been published in international peer-reviewed conference proceedings: ICSEA 2015 [37] and ICSEA 2016 [11] (Appendix A and Appendix B). The latter paper received the "Best Paper" award at the ICSEA 2016 conference and I was invited to write a follow-on journal paper. The paper (Chapter 3) is being submitted to the International Journal on Advances in Software. The former paper led to an observational study presented in Chapter 5, this paper is being submitted to the CSSE&T conference [26]. Table 1 lists the four papers and their status.

Table 1. Scientific Articles

Title / Topic	Event name	Status
Using Cloud Services To Improve Software Engineering Education for Distributed Application Development	ICSEA 2015 [37] (Conference)	Published
A Pattern Language for Application-level Communication Protocols,	ICSEA 2016 [11] (Conference)	Published
CommDP: A Pattern Language for Designing Application-level Communication Protocols. Qualities and Applicability	Iaria journals [38] (Journal)	To be submitted
Improving Computer Science Education Through Cloud Computing: An Observational study	CSEE&T [39] (Conference)	Submitted

CHAPTER 2

A PATTERN LANGUAGE FOR APPLICATION-LEVEL COMMUNICATION PROTOCOLS[4]

2.1. Abstract

Distributed applications depend on application-layer communication protocols to exchange data among processes and coordinate distributed operations, independent of underlying communication subsystems and lower level protocols. Since such protocols are application-specific, developers often must invent or re-invent solutions to reoccurring problems involving sending and receiving messages to meet specific functionality, efficiency, distribution, reliability, and security requirements. This paper introduces a pattern language, called CommDP, consisting of nine design patterns that can help developers understand existing reusable solutions and how those solutions might apply to their situations. Consistent with other pattern languages, the CommDP patterns are described in terms of the problems they address, their contexts, and solutions. The problems and consequences of the solutions are evaluated against four desirable qualities: reliability, synchronicity, longevity, and adaptability for scalable distribution.

2.2. Introduction

At the application level, a distributed system is two or more processes sharing resources and working together via network communications to accomplish a common goal [40][41]. Such systems are ubiquitous in today's Internet-connected world and are

---

[4] Jorge Edison Lascano, Stephen W. Clyde

found in virtually every application domain, such as personal productivity tools, social media, entertainment, research, and business. Even single-user software systems that appear to be non-distributed may in fact communicate with other processes in the background to download updates, track usage statistics, or capture error logs, and are therefore actually distributed systems.

In general, the developers of a distributed system try to increase its overall throughput, reliability, and scalability by hosting data and/or operations on multiple machines, while minimizing network traffic, congestion, and turn-around times. Exactly how they do this depends heavily on the nature and requirements of the application. In some cases, developers may choose to distribute instances of one type of resource, e.g., image files in a peer-to-peer shared photo library. In other situations, developers may group resources such that all instances of a single type are on one server. Still in other cases, developers can take hybrid approaches, distributing certain types of resources among peers and hosting other types on dedicated servers. A closely related design issue deals with the granularity of the distributed resources, i.e., data and operations. From a data perspective, the possible choices range from whole databases to individual records or even individual fields within records. From an operations perspective, the choices range from entire subsystems to atomic operations. With today's programming languages, many developers follow the object-oriented paradigm, encapsulating operations with data and making choices for granularity that range from entire sets of objects to object fragments [42].

Besides deciding on the granularity and distribution of resources (data, operations, or objects), developers often have to consider requirements for security, fault tolerance, maintainability, openness, extensibility, scalability, and dynamic quality of service [2]. The degree to which an application possesses these desirable characteristics is primarily a consequence of architectural design choices, which, in turn, place new requirements on inter-process communications.

The problem is not that existing application-level communications protocols are poorly designed and implemented; rather, the problem is that application developer has to re-invent or re-design them for every new application.

In this paper, we will refer to an exchange among two or more processes for a particular purpose as a *conversation*. A single conversation may be short and simple, like querying a stock's price, or it could be long and complex, like the streaming of a video. The rules that govern a particular type of conversation are a *communication protocol* and a collection of protocols is called a *protocol suite* [43][44].

Application-layer communication protocols (ACPs) are often defined on top of other protocols. For example, the *Hypertext Transfer Protocol* (HTTP), which is an ACP, is defined on top of the *Transmission Control Protocol* (TCP) [40]. Many higher level ACPs, like webservice-based ACPs, are in turn defined on top of HTTP [40]. Section II provides additional background on protocols and protocol suites, as well as a brief discussion on layered communication subsystems.

Because requirements for ACPs can come from an application's (a) functional requirements, (b) architectural design, and (c) use of lower layer protocols, coming up with effective designs can be challenging. Fortunately, the problems that developers are likely to encounter are not uncommon and have known solutions. The key is to capture this knowledge in a way that developers can easily find it and adapt it to a new application. This is precisely what design patterns can do [1].

Unfortunately, design patterns for communication protocols at application layer have yet not been gathered, correlated, and formally organized into a cohesive and thorough collection. To this end, this paper introduces a system of design patterns, i.e., a pattern language, for ACPs, called CommDP. The patterns in CommDP come from a variety of sources and are by themselves not new ideas, as is the case for all newly documented design patterns [45]. Section III provides more background information on design patterns and pattern languages, as well as information about related work.

Since designing ACPs is different from designing executable software, it is necessary to discuss desirable qualities for protocols. Section IV introduces four, namely reliability, synchronicity, longevity, and adaptability for scalable distribution. Section V presents a design pattern template that incorporates these characteristics into the definition of communication problems and the consequences of pattern solutions.

Section VI-A introduces three communication idioms that act as conceptual building blocks for all the ACP patterns in CommDP. We then provide an overview of the ACP patterns in Section VI-B. Additional details for the CommDP patterns are available on-line, at [http://commdp.serv.usu.edu/].

Patterns are rarely used in isolation; instead, developers typically weave multiple pattern instantiations together to create complete solutions [17]. A system of patterns, i.e., a pattern language, not only includes a collection of patterns, but relationships among them that help developers know how they might be effectively combined [46]. Section VI-C provides a digest of these relationships for CommDP. Finally, in Section VII, we summarize the value of CommDP and outline our future research direction.

2.3. Protocols and Protocol Suites

Software and electrical engineers model, design, and implement inter-process communications in layers. Figure 1 shows a simple 5-layer model commonly favored by those who work with IP-based protocols [40][47]. There are several other common models, such as the 7-layer OSI model [48]. A conversation between Process 1 and Process 2 can be discussed at any layer and, for each layer, it must adhere to agree upon protocol(s) for that layer. For example, if Process 1 were a web browser, Process 2 were a web server, and the conversation a simple web-page request, then the application-layer protocol would be HTTP, the Transport-layer protocol would be TCP, and the Network-layer protocol would be IP.

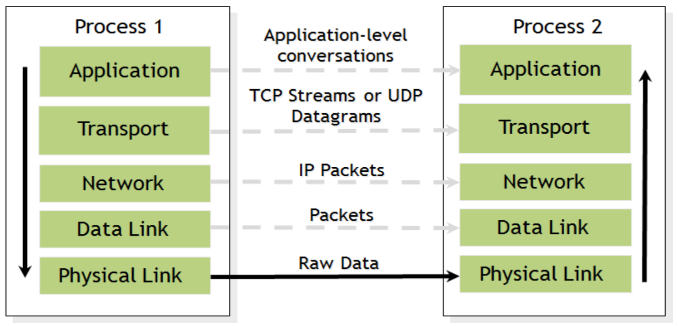


Figure 1. 5-Layer Model for IP-based Communications.

Besides providing a convenient way for discussing protocols, layered models establish a basis for creating substitutable software communication subsystems. Since we are addressing ACPs in this paper, we do not deal directly with design and implementation of these software components. Nevertheless, we assume that appropriate communication subsystems exist at the transport layer for streaming of unstructured data and transmitting datagrams (semi-structured data). Section V relies on this reasonable assumption to define four communication idioms.

At the application layer, a protocol governs why, when, and how processes interact with each other to accomplish a common goal. Specifically, an ACP should define the following:

1. the processes involved in the interaction in terms of the roles they play during a conversation;
2. the possible sequences of messages for valid conversations;
3. the structure of the messages;
4. the meaning of the messages; and,
5. relevant behaviors of the participating processes.

Because processes in a distributed system typically have to communicate with each other for many different tasks, e.g., authentication, resource sharing, and coordination, it is common for a distributed system to require multiple ACPs, i.e., an ACP suite.

2.4. Design Patterns and Pattern Languages

Christopher Alexander et al. defined a pattern as a reusable solution to a reoccurring problem [46]. Kent Beck and Ward Cunningham started to apply the concept of pattern languages to software engineering in 1987 [49], and the idea was later popularized by Eric Gamma et al. with their landmark language of 23 patterns [1]. Since then, pattern languages have been documented for many areas of software engineering, including architectural design [21][50], user-interface design [51], event handling [52] [53] [54], and concurrency [55] [56]. There are even patterns specifically for distributed computing [17], distributed objects [57] [58] [59], communication software [13][60], RESTful and SOAP web services [15], cloud computing [16], and distributed real-time and embedded systems [20]. However, to date, no pattern language has been published specifically for ACPs.

There are two hoped-for benefits of pattern languages that are important to ACP design. First, they create a vocabulary that enables developers to discuss complex ideas in a few words [59]. Second, they allow developers of all experience levels to benefit from expert reusable solutions [61].

2.5. Qualities of Communication Protocols

Like software, ACP suites, as whole, should possess certain desirable qualities that contribute to the overall success of a system. Some of these desirable qualities come directly from the software arena. For example, cohesion is the degree to which the elements of a software component align with a single purpose [62]. Cohesion and its definition can apply almost directly to ACPs, but this is a subject for future research (see Section VII). Another desirable software quality directly applicable to ACPs is modularization. Modularization is the degree to which a system is divided up into independent components [63]. When a system has good modularization, developers do not have to look very far beyond a component to understand it or reason about it. We believe the same to be true for ACPs, but the details of modularization applied to ACPs are also a subject for another paper (see Section VII).

Although we believe cohesion and modularization are important qualities for ACPs, they do not directly help in describing reoccurring communication problems nor are they good discriminators for reusable solutions, because all pattern solutions should, by definition, have good cohesion and modularization. So, we turn our attention to four other qualities with discriminating definitions for ACPs, namely: reliability, synchronicity, longevity, and adaptability for scalable distribution.

2.5.1. Reliability

For an ACP, reliability is the degree to which a process that sends a message as part of a conversation obtains an assurance that the intended recipient(s) received it, entirety and uncorrupted, and reacted as prescribed in the ACP. At the application level, reliability is typically achieved by the recipients returning messages that provide the sender with confirmation that the message was received and/or processed. When such return messages fail to arrive in a timely fashion, reliable ACPs will require the sender to retransmit the original message.

In Section VI, where we present an overview of the ACP patterns in CommDP, we rank each of the patterns in terms of reliability using the following 3-point rubric:

Rank/Criteria

3 The problem (P) addressed by the pattern is primarily concerned with reliability and the solution (S) can make the following guarantees under normal and extreme conditions:

a. The sender can distinguish between successful and failed conversations.

b. The receiver can distinguish between successful and failed conversations.

c. In successful conversations, any process X that sends a message M to process Y, gives a timely assurance to X (in some subsequent message) that Y received M.

d. In successful conversations, for any process X that sends a message M to process Y, if M is supposed to trigger a non-trivial behavior in Y, then X receives a timely assurance that Y successfully handled M.

2 P is concerned with reliability and S can guarantee at least (a) and (c) from above in normal situations.

1 P is not concerned with reliability and S does not limit reliability.

Clearly, there are other conceivable problem/solution criteria for reliability not listed above, such as a reoccurring problem where reliability is a major concern and a solution that does not address it. However, we do not include such meaningless classifications because they would not help classify patterns with expert, reusable solutions.

2.5.2. Synchronicity

In the most general sense, synchronization deals with the coordinated execution of actions in a distributed system and what the state information is necessary for that coordination. This broad definition encompasses, but is not limited to, the common view among programmers that synchronous communications occur when the sender of a message stops and waits for a response from the message receiver [64]. However, this is not the only way to achieve synchronization. Some other common mechanisms are logical clocks [65][66], vector clocks [67][68], vector timestamps [69], optimistic concurrency controls [70], and timing signals.

To evaluate the synchronization requirements for ACPs, we consider: (a) what are the actions that need to be coordinated, (b) where will those actions be executed, and (c) what kind of state information is needed to achieve the desired coordination. A distributed system may perform many different tasks comprised of numerous operations, but rarely all of them have to be fully coordinated. In fact, the more independent the individual operations are, the more a system can maximize concurrency and increase throughput. From a coordination perspective, where the operations take place is actually more important than what the operations do. For example, if all of the actions occur in just one process, then that process may not need to know anything about the state of the other process. Once developers know what operations have to be coordinated and where they will execute, they can consider what local or global state information the coordination logic will need.

To rank synchronicity for ACP patterns, we will use the following definitions:

- C is a conversation involving a closed set of processes, $C.P = \{p_1, \dots, p_n\}$, and a set of messages, $C.M = \{m_1, \dots, m_n\}$, such that $sender(m_i) \in C.P \land receivers(m_i) \subseteq C.P$ for $1 \leq i \leq n$ where $sender(m_i)$ is the process that sent message m_i and $receivers(m_i)$ is the set of processes that received m_i.
- A is a set of operations, $\{a_1, \dots, a_n\}$ that run on $C.P$ and whose execution requires coordination, e.g., ordering, simultaneous execution, etc.
- h(a) is the host process for operation, a, where $a \in A$ and $h(a) \in C.P$

- s(a) is the state information that h(a) needs to coordinate A's execution with the rest of the operations in A.
- H(A) is the set of host processes for all operations in A

Below is an informal 3-point rubric for ranking synchronicity for CommDP patterns using these definitions. We believe that a more rigorous ranking system would have value beyond the categorization of ACP patterns, and its full definition is beyond the scope and purpose of this paper.

Rank/Criteria

3 The problem (P) addressed by the pattern deals with situations where $|\mathrm{H(A)}| > 1$ and the solution (S) can guarantee that for all a $\in$ A, h(a) receives s(a) via messages, $\mathrm{m_i} \in \mathrm{C.M}$, in time to do the prescribed coordination.

2 P deals with situations where $|\mathrm{H(A)}| = 1$ and S can guarantee that for all a $\in$ A, h(a) receives s(a) via messages, $\mathrm{m_i} \in \mathrm{C.M}$, in time to do the prescribed coordination.

1 P is not concerned with synchronicity, e.g., $|\mathrm{A}| = 0$, and S does not limit synchronicity.

2.5.3. Longevity

Longevity is the degree to which an ACP can support long-running conversations caused by long-running operations. The primary problem for conversation with long-running operations is that there could be huge span of times when processes are uncertain of each other's states. Consider a simple request/reply conversation where some process A sends a request to B, but B takes a long time to execute the requested operation and sends back a reply. While waiting for the reply, process A does not know if B received the request, has failed, or is just taking a long time. ACPs that support long-running operations include mechanisms for exchanging state information independent of results.

We rank the longevity for ACP patterns according to the following 3-point rubric:

Rank/Criteria

3 The problem (P) addressed by the pattern is primarily concerned with long-running conversations and the solution (S) can guarantee the following in successful conversations:

   a. Participants made aware of each other's states in periodically.
   b. Each participant in the conversation can detect when other participants are no longer available or accessible.

2 P is concerned with long-running conversations and S provides for (a).

1 P is not concerned with long-running conversations and S does not limit longevity.

2.5.4. Adaptability for Scalable Distribution

ACPs can support scalability by providing location transparency and/or replication transparency [69], and by allowing resources (data, operations, or objects) to be distributed across multiple hosts. To understand location and replication transparency, consider a website with a large number of resources. It can support scalability by placing the various resources on an expandable collection of backend servers and use a front-end server to distribute requests from browsers. If the browser does not need to know where a resource is actually located, then the system supports location transparency. Similarly, as traffic increases, the system could replicate resources across multiple backend servers. If the client does not have to know that replicas exist, then the system supports replication transparency. Both location and replication transparency simplify scalability.

Another technique for supporting scalability is allowing complex resources to be broken up into smaller resources and distributed across multiple servers. One approach for doing this is to untangle cross-cutting concerns, like security or logging, from complex operations and host these pieces of functionality on proxies [71][69].

Here is a simple rubric for the adaptability for scalable distribution.

Rank/Criteria

3 The problem (P) addressed by the pattern is primarily concerned about scalability or the distribution of action or resources and the solution (S) can provide two or more of following:

   a. Location transparency for shared resources distributed across multiple hosts
   b. Load balancing with shared resources replicated across multiple hosts
   c. Untangling of cross-cutting concerns into separate actions

2 P is concerned with scalability or distribution of resources and S provides at least one (a), (b), or (c).

1 P is not concerned with scalability or distribution and S does not limit them.

2.6. Template for Communication-protocol Patterns

To document the patterns in CommDP, we have developed a template, loosely based on the way Gamma, Helm, Johnson and Vlissides documented their patterns [61], referred to here as the GoF template. The main goal is to keep the documentation as simple as possible, while still capturing the details of the pattern. Following are the elements of CommDP pattern template.

2.6.1. Name

As with the GoF template, the name uniquely identifies the pattern. Since the name will become part of the vocabulary for the pattern language, it is important that it captures the essence of the pattern, distinguishes it from other patterns, and is as concise as possible.

2.6.2. Intent

The intent is an abstract for the pattern. It summarizes the problem, the context, and the solutions, particularly in terms of reliability, synchronicity, longevity, and adaptability for scalable distributes.

2.6.3. Description

The description consists of three subsections that explain the problem, context, and solution. The problem subsection relates closely to the Motivation part in the GoF template, in that it explains the nature of the reoccurring communication-protocol design problem. This subsection should highlight the problem's need for reliable communications, synchronization, long-running conversations, or scalable distribution. The context subsection is like the Applicability in the GoF template, capturing information when the pattern may or may not be applicable and assumptions about distributed systems in which the communications will take place. The solution is analogous to the Structure in the GoF template. It focuses on the describing protocol design ideas and how they can be adapted.

2.6.4. Consequences

As with the GoF template, the consequences are important part of CommDP pattern definitions because developers will use them to determine if the pattern is a good fit for a particular situation. The consequences of CommDP patterns are described in terms of the qualities discussed in Section 4. The rankings provide a general classification, and pros explain the consequences in more detail.

2.6.5. Known Uses

Like the GoF template, this part references known instances of the pattern in production systems.

2.6.6. Aliases and Related Work

The section combines two elements of the GoF template by the similar names.

2.6.7. References

This section contains a bibliography for the citations made elsewhere in the pattern definition.

2.7. CommDP

A design pattern is composed of a set of patterns and idioms that are used together to solve a design engineering problem.

2.7.1. ACP Idioms

Before launching into a description of CommDP's patterns, it is important to first introduce three fundamental building blocks for all ACPs: point-to-point send, multicast, and broadcast. These are idioms instead of patterns because their usage depends on the lower layer communication protocols and because, by themselves, they do not address the qualities discussed in Section 4.

A *point-to-point* send is the transmission of a single message from one process to another, such as a message sent over a TCP connection or via a UDP datagram. An underlying communication subsystem may provide some reliability relative to the transmission but, at the application-level, a single message does not allow the sender to know if the receiver processed the message or anything about the receiver's state, nor does it help with longevity or adaptability for scalable distribution.

A *multicast* send is the transmission of a single message to a set of receiving processes [72]. It can be implemented at virtually any layer in communication hierarchy, including the physical layer. Mechanisms for identifying the group of receiving processes vary from sender determined to receiver subscriptions. By themselves, multicast are idioms for ACPs. The same is true for *broadcasts*, which also transmit messages to multiple receivers [72].

2.7.2. ACP Patterns

Table 2 lists the nine patterns currently in CommDP, along with their rankings from their consequences relative to the characteristics discussed in Section 4 (R=Reliability, S=Synchronicity, L=Longevity, and A=Adaptability for Scalable Distribution). Their full definitions are available on [http://commdp.serv.usu.edu].

The *Request-Reply* pattern is undoubtedly the most common. It addresses the problem where a process, A, needs to access or use shared resources in another process, B, with a reasonable degree of reliability and synchronicity. The solution consists of A sending B a message (i.e., a request) and B sending back a message (i.e., a reply) after processing the request, as you can see in Figure 2. For A, this simple mechanism provides a modest level of reliability and synchronization, because the reply proves that B received the request and can provide relevant information about B's state. Furthermore, if A does not receive a reply within a specific amount of time (i.e., a timeout), it can resend the request. It can continue to timeout and retry until it eventually receives a reply or it exceeds some maximum number of retries. This "timeout/retry" behavior is the essence of the request-reply pattern.

Table 2. CommDP Patterns

Name	**Consequences**			
	R	**S**	**L**	**A**
Request-Reply	2	2	1	1
Request-Reply-Acknowledge	3	3	1	1
Idempotent Retry	3	1	1	1
Intermediate State Messages	3	3	3	1
Second Channel	1	3	3	1
Front End	1	1	1	3
Proxy	1	1	1	3
Reliable Multicast	3	3	1	2
Publish-Subscribe	2	1	1	3

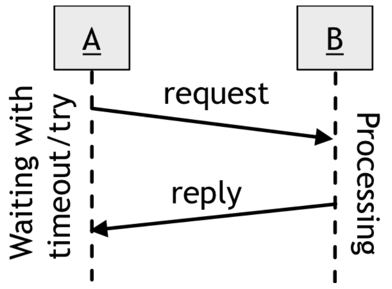


Figure 2. Request-Reply Message Sequence.

The *Request-Reply-Acknowledge* pattern extends this solution with a third message (an acknowledgement) that A sends to B after receiving the reply, and gives B a timeout/retry behavior with respect to its sending of the reply and waiting for an acknowledgement, see Figure 3. This pattern is useful in situations were significant processing may occur on A after receiving the reply or when it is problematic for B to reprocess duplicated requests caused by A's timeout/retry behavior. With this pattern, instead of reprocessing a duplicate request, B can simply cache its replies and resends them to A when necessary. The acknowledgement tells B that A has received the reply and, thus, can remove it from its cache. This pattern offers more reliability and synchronization than request-reply, but at the cost of an additional message.

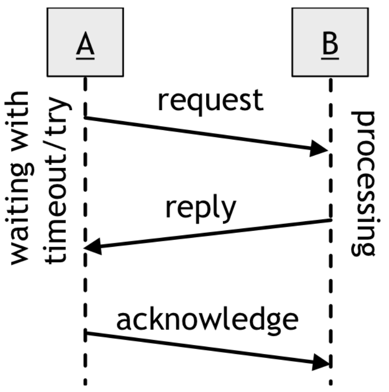


Figure 3. Request-Reply-Acknowledge Message Sequence.

The *Idempotent Retry* pattern [73] captures a different solution to the problem of processing duplicate requests. Like Request-Reply, its solution consists of A sending a request to B with a timeout/retry behavior and B sending a reply back to A. But, unlike Request-Reply, the semantics of the protocol dedicate the processing of the request must be idempotent. This pattern applies to situations where the requested processing is relatively light, i.e., less expensive than caching replies.

The next pattern, *Intermediate State Message*, is also similar to Request-Reply, but addresses the problem of long-running conversations due to request actions taking substantial amounts of time to complete. To solve this problem, it has B send A one or more intermediate messages that reflect its current state. For example, B may send a message immediately after receiving the request to let A know that it got the request, another message when the processing is 10%, another at 20% complete, and so on. Each intermediate message provides state information about B, which improves synchronization in the presence of time-consuming actions, see Figure 4.

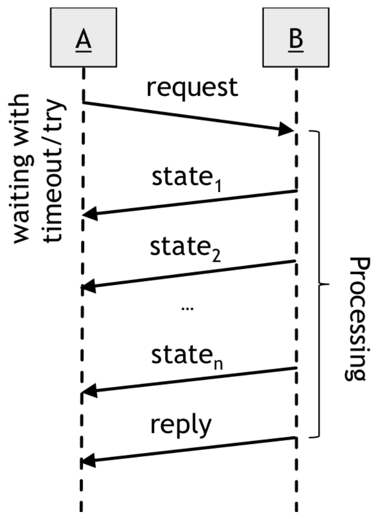


Figure 4. Intermediate State Message Sequence.

The *Second Channel* pattern is also for situations involving long-running conversations, but ones dominated by significant amounts of data transfers instead of time-consuming actions. Because the large data transfers can delay intermediate state messages, this pattern's solution suggests opening a second communication channel between A and B that is dedicated to data transfer, leaving the original communication channel available for intermediate state or control messages, as it is shown in Figure 5. The File Transfer Protocol (FTP) and its variations are classical examples of this pattern[47][72].

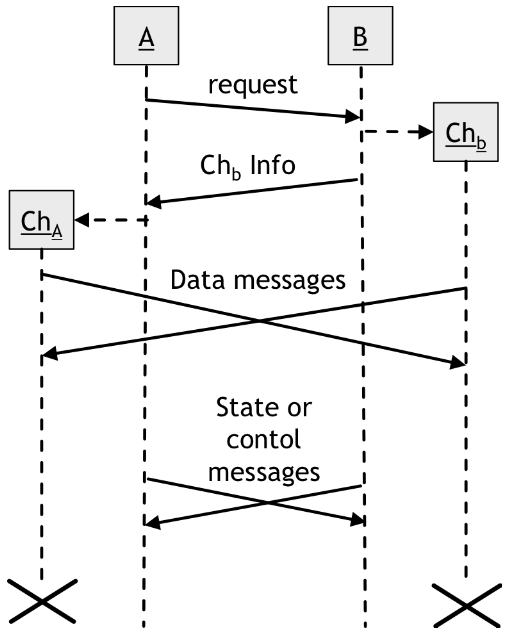


Figure 5. Second Channel Message Sequence.

The *Front End* pattern addresses the problems of making the location of shared resource transparent to the client, allowing the number of resources to change dynamically. It has a resource client send requests to a front-end process that automatically redistributes them to appropriate resource managers, B processes. After processing the request, a resource manager replies back to the client directly, for a graphic description of this pattern, you can see Figure 6. The front-end process can use a variety of criteria to decide how to redistribute requests, including request type, resource type or identity, and resource manager load. By itself, this pattern's primary focus is on the distribution and scalability of resources.

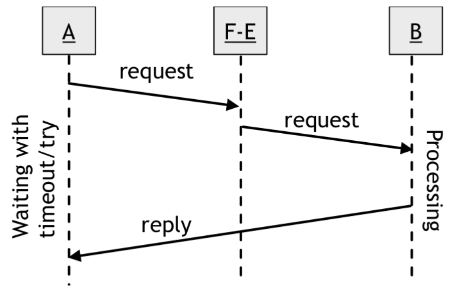


Figure 6. Front End Message Sequence.

Like the Front End, the *Proxy* pattern, presented in Figure 7, introduces a process between a resource client and a resource manager. However, the intermediate process, called a proxy, serves other functional purposes besides re-distribution of the requests, for example it may provide authentication, access control, audit logging, and data transformation functionality. Also, the resource manager returns replies through the proxy to client, completely isolating the client from the resource manager.

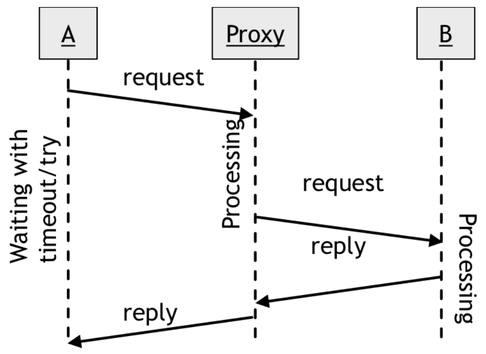


Figure 7. Proxy Message Sequence

The *Reliable Multicast* pattern builds on the multicast idiom to provide reliability and synchronization among a group of processes. Its solution is a protocol that starts with a process A sending a request message to a group of process, B={b_1, .., b_n}. Each process b_i sends a reply back to A when it receives the request and is ready to process it. After A receives reply from all B processes, then A will multicast a go-ahead message back out to all B message indicating that they can proceed with the processing of the request, shown in Figure 8. In this way, the execution of the request is synchronized among all of the B processes. If A fails to receive a reply from every B process, it can resend the request to some or all of them until it gets a reply from all of them or terminates the conversation as failed. This pattern focuses on providing strong reliability and synchronization, but can also help with scalable distribution of resources.

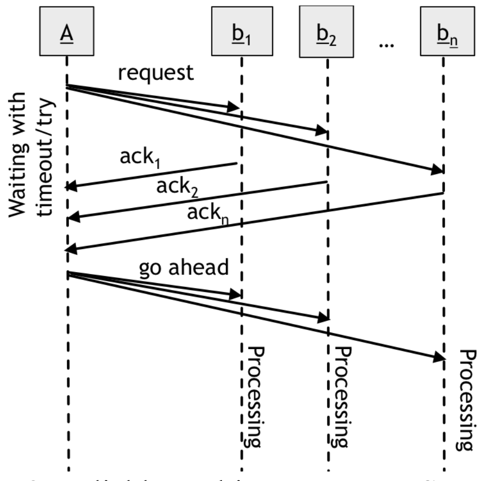


Figure 8. Reliable Multicast Message Sequence.

Finally, the *Publish-Subscribe* [17] pattern is a powerful mechanism for decoupling message senders (publisher) from message receivers (subscribers). With this pattern, an intermediate process acts as a store-and-forward buffer for message transmission with the capabilities for managing subscribers and delivering individual message to multiple subscribers.

2.7.3. CommDP: Pattern Relationships and Composition

Patterns are rarely used in isolation; instead, developers combine their solutions to solve complex problems. Virtually any of the CommDP patterns could be combined with any other pattern, but the more useful combinations are ones that have complimentary characteristics, like Request-Reply with Second Data Channel or Request-Reply Acknowledge with Front End.

To ensure that the CommDP pattern set was as minimal as possible, we did not include in any pattern in CommDP that was simply an aggregation of two or more patterns. For example, there is a common type of distributed system that deals with information flow and processing. In such systems, a process A might send a request to B through a series of intermediate proxy-like processes that transform or augment data in request on its way to B. At each intermediate step, a reply is sent back to A, informing it of the message's process. Eventually, when the transformed message arrives at B and processes it, then B sends a final reply message back to A. This particular solution offers good reliability, synchronization, longevity, and adaptability to scalable distribution, but it is actually just a composition of the Proxy pattern (applied perhaps multiple times) and the Intermediate State Message pattern.

2.8. Summary and Future Work

CommDP pulls together reusable solutions to reoccurring design problems with ACPs, filling a much needed gap in the knowledge base for developers of distributed systems. We have characterized the nature of the problems that the CommDP patterns address and the consequences of their solution in terms of four desirable qualities, namely: reliability, synchronicity, longevity, and adaptability for scalable distribution. These qualities are both instructive and discriminating, in that they can help a developer understand the solutions and choose the most appropriate solution for a given situation. However, more work needs to be done to formalize these qualities and to solidify their sufficiently and completeness relative communication-protocol design. So this is one of our research group's immediate goals.

We also hope to investigate other qualities, like cohesion and modularization that might be valuable for protocol design even if they are not good discriminators for design patterns. Being able to reason about assess, and teach these qualities more formally will help developers create better distributed systems.

Finally, over time, we hope the expand the patterns in CommDP, without adding any that are just compositions of existing patterns, to encompasses a boarder range of reusable solutions for ACPs.

CHAPTER 3

COMMDP: A PATTERN LANGUAGE FOR DESIGNING APPLICATION-LEVEL COMMUNICATION PROTOCOLS QUALITIES AND APPLICABILITY[5]

3.1. Abstract

This work is an extension to the paper "A Pattern Language for Application-level Communication Protocols" [1], where we introduced the *Application level Communication Protocols Design Pattern Language* (CommDP). The present article aims to define a complete description of the patterns, show examples of their applications, and analyze their pertinence with the communication protocols' qualities, namely, reliability, synchronicity, longevity, and adaptability for distribution, i.e., to analyze non-functional requirements of communication protocols and propose an early quality model for Application-level Communication protocols.

3.2. Introduction

With the current expectations for "connected" applications, many software developers today are concerned with designing reliable, secure, efficient, and scalable *application-level communication protocols* (ACPs). In general, a communication protocol is a set of rules that governs a particular type of conversation among communicating entities. ACPs are high-level communication protocols for application-layer components and are typically built on top of transport, session, presentation, or even other application protocols [1] [2]. For example, many web-based applications have ACPs that are built on

[5] Jorge Edison Lascano, Stephen W. Clyde

top of the *Hypertext Transfer Protocol* (HTTP), which itself is an ACP, and HTTP is built on top of the *Transmission Control Protocol* (TCP), which is a transport protocol [3]. A collection of related protocols, like all those used by a single distributed application, form a *communication protocol suite* [4][5]. It is common for an application's protocol suite to have dozens of ACPs, each with its own specific purpose and conversation-governing rules. Section 3.3 provides some additional background on these and other foundational concepts, such as quality models.

This paper aims to capture existing design expertise for ACPs in design patterns, which collectively form a pattern language [6], using a consistent and appropriate pattern-documentation template. Specifically, this article updates and extends the initial work in this area, presented in a paper at ICSEA 2016, titled "A Pattern Language for Application-level Communication Protocols" [1]. We have also setup a wiki, available on-line[6], for collaborative work on its ongoing extension and evolution.

To help developers understand the consequence of using any CommDP design pattern, this paper describes four communication-protocol qualities: reliability, synchronicity, longevity, and adaptability for scalable distribution, and then evaluates each pattern respective to its potential impact on these qualities. This can help developers make informed choices about which pattern(s) to use in any given situation. Section 3.4 explains the four communication qualities and introduces a quality model for classifying CommDP patterns relative to these qualities.

To realize the potential benefit of a pattern language, it is important to document

[6] http://commdp.serv.usu.edu/

the essential concepts of each pattern, the consequences of using the pattern, examples, and other relevant information using a consistent documentation template. Section 3.5 explains CommDP's documentation template and why it is necessary and appropriate for ACPs. Section 3.6 contains the definitions for the eleven patterns currently in CommDP and introduces several patterns currently being considered for inclusion in CommDP. Finally, Section 3.7 provides as summary and ideas for future work.

3.3. Background

3.3.1. Communication Protocols

Software engineers model, design, and implement inter-process communications in layers. Figure 9 shows a simple 5-layer model commonly favored by those who work with IP-based protocols [3][7]. Another common model is the 7-layer OSI model [8]. As illustrated in Figure 9, a conversation between Process 1 and Process 2 can be discussed at any layer. The components at each layer must adhere to agree upon protocol(s) for conversations to take place. For example, if Process 1 were a web browser and Process 2 were a web server and they wanted to have a conversation involving the request and delivery of a web-page, the application-layer component would need to agree on a protocol for that purpose. The *Hyper-Text Transfer Protocol* (HTTP) is an ACP specifically designed for this purpose. It is built on top of a transport-layer protocol, called Transport-Communications Protocol (TCP), which "streams" the request for a webpage to Process 2 and the desired webpage is sent back to Process 1 in a reliable way. TCP is in turn built on top of the network-layer protocol called Internet Protocol (IP), which packetizes the stream and routes through the network.

Besides providing a convenient way for discussing protocols, layered models establish a blueprint for creating software communication subsystems. Since we are addressing ACPs in this paper, we do not deal directly with design and implementation of these software components.

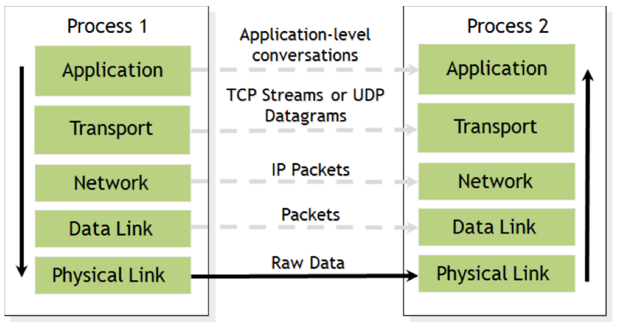


Figure 9. Layer Model for IP protocols.

At the application layer, a protocol governs why, when, and how processes interact with each other to accomplish a common goal. Like all communication protocols, they must prescribe message syntax, semantic, synchronization, and error recovery [4]. Specifically, an ACP should define the following: the processes involved in the interaction in terms of the roles they play during a conversation, the possible sequences of messages for valid conversations, the structure of the messages, the meaning of the messages, and relevant behaviors of the participating processes. Although ACPs can be implemented in any type of application that needs to attain some type of conversation, they are commonly used to develop distributed applications.

Because requirements for ACPs can come from an application’s (a) functional requirements, (b) architectural design, and (c) use of lower layer protocols, coming up with application-specific ACPs designs that are reliable, secure, efficient, and scalable can be challenging. Fortunately, the problems that developers are likely to encounter are not uncommon and have known solutions. The key is to capture existing expertise in a way that developers can easily find it and adapt it to a new application. This is precisely what design pattern languages [6], like CommDP, aim to do.

3.3.2. Patterns and Pattern Languages

Christopher Alexander et al. defined a pattern as a reusable solution to a reoccurring problem [9]. Kent Beck and Ward Cunningham started to apply the concept of pattern languages to software engineering in 1987 [10], and the idea was later popularized by Eric Gamma et al. with their landmark language of 23 patterns [6]. Since then, pattern languages have been documented for many areas of software engineering, including architectural design [11][12], user-interface design [13], event handling [14][15][16], and concurrency [17]-[18]. There are even patterns specifically for distributed computing [19], distributed objects [20][21][22], communication software [23][24], RESTful and SOAP web services [25], cloud computing [26], and distributed real-time and embedded systems [27]. However, to date, no pattern language has been published specifically for ACPs.

There are two hoped-for benefits of pattern languages, in general. First, a pattern language can create a vocabulary that enables developers to discuss complex ideas in a few words [22]. This is particularly useful in ACP design because of the variations in terminology that already exists. For example, request-response [28], request reply [29], and request acknowledge [25] can be used to express a two-way or a one-way communication indistinctly. Second, a pattern language allows developers of all experience levels to benefit from expert reusable solutions [30] and to apply those ideas to improve the quality of their systems in predictable and measurable ways.

3.3.3. Quality and Quality Models

ISO 25010 [39] is a hierarchical structure that describes the quality of a software products based on desired qualities, sub-qualities, and attributes. Software engineers use formal or informal metrics to measure an attribute to assess the degree to which a software system possesses that attribute. Then, they aggregate their assessments of the individual attributes to conclude the presence (or lack) of sub-qualities and ultimately, the desired qualities.

Similarly, the quality of communication protocols could be determined through quality, sub-qualities, attributes, but these would be necessary different from software systems since communication protocols not self-contained computational mechanisms, but guidelines for how multiple entity interaction.

Several authors have dealt with quality issues related for protocols, including: formal methods in communication protocol design [31], techniques and principles that lead to good protocol designs [32] [34], best practices for implementing protocols [33], and performance issues [48] [49] [50]. However, none of them discuss or present a model to assess ACP qualities.

G. Coulouris et al., wrote “It is of high interest to be able to estimate, or even measure the quality of a system under construction” [35]. In fact, in any engineering field, it is generally believed that you can not improve what you do not measure.

Software quality metrics are used to identify strengthen or weakness in a software system and can therefore become a basis for improvement. Developers should not only aim to achieve the required functionality of products, but do so in a way that it possesses desirable qualities like reliability, security, efficiency, scalability, maintainability, reusability, extensibility, portability, and testability [36]. For example, measuring Coupling and Cohesion (C&C) can help decrease the possibility of a fault or a defect appearance and maintenance if applied early in the development process [35]. Also complexity quality metrics are used to reduce defects [37].

Quality models are composed by desirable characteristics, .i.e., “qualities”, that relate to “static properties of software and dynamic properties of the computer systems” [36, p. 25] They provide a set of quality characteristics that helps verify if the intended requirements of a software product are met. ISO/IEC 25010 [39, p. 25010], a standard software quality model, proposes the following set of qualities for system evaluation: functionality, maintainability, usability, reliability, security, and performance efficiency. However, these qualities pertain to software that produces executable code and not to protocols that govern conversations being executing entities. Section 3.4 introduces a quality model specifically for ACP and ACP patterns.

3.3.4. Design Patterns

Distributed applications depend on application-layer communication protocols to exchange data among processes and coordinate distributed operations, independent of underlying communication subsystems and lower level protocols. Since such protocols are application-specific, developers often must invent or re-invent solutions to reoccurring problems involving sending and receiving messages to meet specific functionality, efficiency, distribution, reliability, and security requirements. Here is where a pattern language is important for ACP design [1]. First, it creates a vocabulary that enables developers to discuss complex ideas in a few words [22]. Second, they allow developers of all experience levels to benefit from expert reusable solutions [30].

3.4. The CommDP Quality Model

As with software systems at large, when designing ACPs, designers try to meet certain general and application-specific objectives. The general objectives include minimizing communication overhead, minimizing response times, and maximizing throughput. Regardless of the application, these are viable objectives for any ACP. Of course, how much effort or expense developers put forward to satisfy them will still depend on the available time and budget. The application-specific objectives fail into four broad categories: reliability, synchronicity, longevity, and adaptability for scalable distribution. In other words, depending on application requirements, designers may need to create ACPs that can tolerate certain kinds of failures, e.g. lost messages; ensure that processes can achieve a degree of synchronization for certain operations; accommodate long-running operations; or allow for scalability with respect to users or resources. From this point on, we will reference these four application-specific objectives as the RSLA qualities.

To help ACP designers reuse design expertise embedded in existing ACPs, we propose a quality model, called the *CommDP Quality Model*, that possesses the following features:

1. Enabling the assessment of an ACP's pattern towards the RSLA qualities
2. Allowing for the ranking of requirements relative to RSLA qualities
3. Enabling the assessment of the degree an ACP supports the RSLA qualities

Feature #1 means that ACP patterns can be classified according to how they help with the RSLA qualities. Feature #2 means that designers can characterize the problem they are working on. Therefore, with the quality model, designers should be able to find patterns or combinations of patterns that will contribute to a meaningful solution for their specific application. Feature #3 means that quality model should allow designers to verify their completed ACP for support of the RSLA qualities, to the desired degrees. Figure 10 provides an overview of CommDP Quality Model. Specifically, it shows how each RSLA quality is sub-divided into one or more attributes. Sections 3.4.1-3.4.4 describe the RSLA qualities in more detail, as well as rubrics for ranking the attributes on a three-point scale (0-2), where a 0 (zero) means that an ACP or design pattern is not concerned with that attribute, while a 1 or 2 means that either contributes to or fully achieves the attribute.

Very rarely would an ACP possess all the attributes and none of these attributes is essential for every ACPs. Therefore, choosing an appropriate pattern or combination of patterns for the ACP is matter of matching the need for specific attributes, stemming from application requirements, with levels of the attributes supported by the patterns.

3.4.1. Reliability

For an ACP, reliability is the degree to which a process can send a message to one or more intended recipient(s) and know that those recipient(s) received it, in its entirety and uncorrupted, and that they reacted as expected. An ACP typically achieves reliability by having the recipient(s) return another message to the sender, confirming that the original message was received and processed.

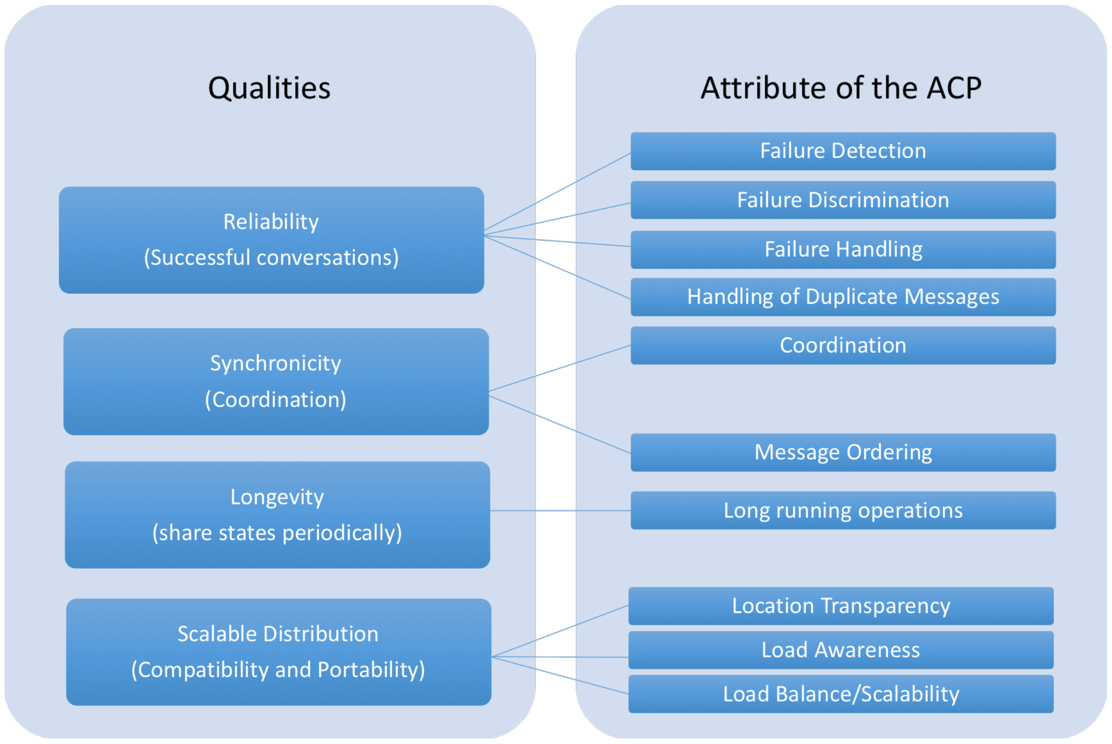


Figure 10. ACP Quality Model.

If the sender fails to receive the confirmation message in a timely fashion, a reliable ACP will typically require the sender to retransmit the original message or some other message then reverts the conversation to a previous state. To accomplish this, an ACP must include specifications for how a sender can detect and discriminate communication failures, and either retransmit previous messages or alternative messages as needed. It also requires an ACP to specify how receiving processes should handle duplicate messages. Therefore, we decompose reliability into four independent attributes: detection of failures, discrimination of failure types, failure handling, and accommodating duplicate messages. Table 3 shows a rubric for ranking requirements, patterns, and ACPs for each of these attributes. Section 3.6.11 shows assessments of these attributes for each of the CommDP patterns.

Table 3. Ranking Scale for Reliability Attributes

Ranking scale	Reliability attribute/ranking description
Detection of failures	
0	No conversation participant (process) needs to detect a failure
1	One of the conversation participants can
2	Multiple conversation participants can
Discrimination of Failure Types	
0	There is no need to discriminate between different types of failures
1	Participant(s) can discriminate between action, process, or transmission failures
2	Participant(s) can discriminate between action, process, or transmission failures
Failure Handling	
0	The protocol does not deal with the reverting a conversation to a previous state and resending of messages
1	Allows at least one process to back up to the last state and resending one message
2	Allows processes to back up to any of a set of previous states and resending messages accordingly
Handling Duplicate Messages	
0	Does not deal with duplicate messages
1	The protocol requires at least one process to detect duplicate messages and handle them appropriately, but does not deal with efficiency issues
2	The protocol allows for duplicate messages, but tries to minimize their occurrence and decrease negative consequences of duplicate messages

3.4.2. Synchronicity

In the most general sense, synchronization deals with the coordinated execution of actions in a distributed system and what the state information is necessary for that coordination. This broad definition encompasses, but is not limited to, the common view among programmers that synchronous communications occur when the sender of a message stops and waits for a response from the message receiver [40]. However, this is not the only way to achieve synchronization. Some other common mechanisms are logical clocks [41][42], vector clocks [43][44], vector timestamps [45], optimistic concurrency controls [46], and timing signals.

To evaluate the synchronization requirements for ACPs, we consider: (a) what are the actions that need to be coordinated, (b) where will those actions be executed, (c) what kind of state information is needed to achieve the desired coordination, and (d) the pertinence or not of the order of the delivered messages. A distributed system may perform many different tasks comprised of numerous operations, but rarely all of them have to be fully coordinated. In fact, the more independent the individual operations are, the more a system can maximize concurrency and increase throughput. From a coordination perspective, where the operations take place is actually more important than what the operations do. For example, if all of the actions occur in just one process, then that process may not need to know anything about the state of the other process. Once developers know what operations have to be coordinated and where they will execute, they can consider what local or global state information the coordination logic will need. Ultimately, the two dimensions of synchronicity are coordination, and ordering of messages. Table 4 shows the rubric to evaluation these two attributes.

Table 4. Ranking Scale for Synchronicity Attributes

Ranking scale	Synchronicity attribute/ranking description
	Coordination
0	No coordination is needed among the participants
1	One process can know something about the state of another process; the protocol supports actions on one process and guarantees that message(s) which need to arrive before those actions take place do so.
2	Multiple processes can know something about the state of other processes; the protocol supports actions being performed on multiple processes and guarantees that messages which need to arrive before those actions take place do so.
	Ordering of messages
0	Messages ordering is not a concern
1	The protocol allows local ordering of message processing
2	The protocol allows global ordering of message processing

3.4.3. Longevity

Longevity is the degree to which an ACP can support long-running conversations caused by long-running operations. The primary problem for conversation with long-running operations is that there could be huge span of times when processes are uncertain of each other's states. Consider a simple request/reply conversation where some process A sends a request to B, but B takes a long time to execute the requested operation and sends back a reply. While waiting for the reply, process A does not know if B received the request, has failed, or is just taking a long time. ACPs that support long-running operations include mechanisms for exchanging state information independent of results. Longevity main and only attribute is its support for long-running operations, see Table 5 for its ranking scale. CommDP's patterns are analyzed in function of long running operations in section 3.6.11.

Table 5. Ranking Scale for Longevity Attributes

Ranking scale	Longevity Attribute/Description
Support for long running operations	
0	Long running conversation are not a concern
1	One participant is aware of the another's state throughout a long running operation
2	All Participants are aware of each other's states throughout a long running operation

3.4.4. Adaptability for Scalable Distribution

ACPs can support scalability by providing location transparency and/or replication transparency [45], and by allowing resources (data, operations, or objects) to be distributed across multiple hosts. To understand location and replication transparency, consider a website with a large number of resources. It can support scalability by placing the various resources on an expandable collection of backend servers and use a front-end server to distribute requests from browsers. If the browser does not need to know where a resource is actually located, then the system supports location transparency. Similarly, as traffic increases, the system could replicate resources across multiple backend servers. If the client does not have to know that replicas exist, then the system supports replication transparency. Both location and replication transparency simplify scalability.

Another technique for supporting scalability is allowing complex resources to be broken up into smaller resources and distributed across multiple servers. One approach for doing this is to untangle cross-cutting concerns, like security or logging, from complex operations and host these pieces of functionality on proxies [47][45]. An ACP is said to be fully scalable if it provides attributes such as location transparency, load awareness and load balancing, Table 6 shows the ranking scale for these three attributes. While Table 10 presents their analysis in respect to the CommDP patterns.

Table 6. Ranking Scale for Adaptability Attributes

Ranking scale	Adaptability Attribute/Ranking Description
Location transparency	
0	Location transparency is not a concern
1	Resource users do not have to know the network addresses of the resource managers hosting the desired resources.
2	Resources user do not have to know which resource manager is hosting the desired resources
Load awareness	
0	The protocol does not include mechanisms for communicating current load information
1	The protocol requires one or more process to estimate load information of other processes
2	The protocol allows one or more processes to obtain timely load information for other processes
Load balancing and scalability	
0	Scalability and load balancing is not a concern
1	The protocol supports load balancing across existing process (known potential participants)
2	The protocol supports the dynamic addition / removal of resources and conversation participants

This quality model helps developers select the most appropriate pattern to apply in their implementation of the needed communication protocols. The understanding of

CommDP patterns is documented using a pattern template defined in the following section.

3.5. CommDP Template

In [1] CommDP, which is a pattern language that collects the necessary knowledge to solve reoccurring problems of ACPS' design. These patterns are documented consistent with other pattern languages, the CommDP patterns are described using a template in terms of the problems they address, their contexts, and solutions. This template is loosely based on the way Gamma, Helm, Johnson and Vlissides documented their patterns [30], referred to here as the GoF template. The main goal is to keep the documentation as simple as possible, while still capturing the details of every pattern. Whereas, the name, intent and description should be sufficient to give the reader a high-level understanding of the pattern. Following are the descriptions of the CommDP pattern template's elements.

3.5.1. Name and Overview

As with the GoF template, the name uniquely identifies the pattern. Since the name will become part of the vocabulary for the pattern language, it is important that it captures the essence of the pattern, distinguishes it from other patterns, also it should be descriptive, concise and precise. Additional to the name, this section introduces a brief description of the pattern's context in terms of the participants and the messages involved in the conversation.

3.5.2. Intent

The intent is an abstract for the pattern. It summarizes the problem, the context, and the solutions, particularly in terms of reliability, synchronicity, longevity, and adaptability for scalable distributes.

3.5.3. Description

The description should explain the problem, context, and solution of the pattern. The problem explanation is closely to the *Motivation* part in the GoF template, in that it explains the nature of the reoccurring communication-protocol design problem. It should highlight the problem's need for reliable communications, synchronization, long-running conversations, or scalable distribution.

The context part of the description is like the *Applicability* section in the GoF template, capturing information about when the pattern may be applicable and assumptions about the distributed systems in which it may be used.

The solution is analogous to the *Structure* component in the GoF template. It focuses on the describing protocol design ideas and how they can be adapted. UML Sequence diagrams are a useful tool for example message sequences prescribed by the solution. Like more the solution captures in general design patterns, a CommDP solution is just a partial design; it only specifies certain aspects of a protocol definition that address the reoccurring problem in the specified context. A solution may leave significant parts of the protocol design unspecified.

3.5.4. Consequences

As with the GoF template, the consequences are important part of CommDP pattern definitions because developers will use them to determine if the pattern is a good fit for a particular situation. The consequences of CommDP patterns are described in terms of the characteristics discussed in section 3.4. The proposed model provides a general classification, and explains the consequences of the pattern in more detail.

3.5.5. Known Uses

Like the GoF template, this part references known instances of the pattern in production systems, i.e. protocols or systems that implement the specific pattern.

3.5.6. Aliases and Related Work

This section lists aliases that exist for the pattern. It combines two elements of different pattern languages by similar names or intents, they may or may not pre-exist. This section also contains relevant books, pattern languages, or articles where the pattern has been used to solve specific reoccurring problems by different authors.

3.5.7. Examples of Applications

This section shows actual examples of how the pattern is being or can be applied or adapted. It describes the example, its fitness according to the proposed model and a scenario describing the messages and their sequences.

Different authors have defined pattern languages related to distributed or client-server applications, namely, design patterns for communication service design patterns service[51], pattern oriented software architecture [19], patterns of enterprise application architecture [52]. Nevertheless, none of them deal with the design of communication protocols at any level of the layer model (section 3.3.1).

3.6. CommDP Patterns

CommDP can help developers understand existing reusable solutions and how those solutions might apply to their situations. Before launching into a description of CommDP's patterns, it is important to first introduce three fundamental building blocks for all ACPs: point-to-point send, multicast, and broadcast. These are *Idioms* instead of patterns because their usage depends on the lower layer communication protocols and because, by themselves, they do not address the qualities discussed in 3.4.

A *point-to-point* send is the transmission of a single message from one process to another, such as a message sent over a TCP connection or via a UDP datagram. An underlying communication subsystem may provide some reliability relative to the transmission but, at the application-level, a single message does not allow the sender to know if the receiver processed the message or anything about the receiver's state, nor does it help with longevity or adaptability for scalable distribution.

A *multicast* send is the transmission of a single message to a set of receiving processes [2]. It can be implemented at virtually any layer in communication hierarchy, including the physical layer. Mechanisms for identifying the group of receiving processes vary from sender determined to receiver subscriptions. By themselves, multicast are idioms for ACPs. The same is true for *broadcasts*, which also transmit messages to multiple receivers [2].

The patterns currently in CommDP are grouped into three categories. *2 participant*, this means that there are only two processes involved in the communication. The second category is *3+participant*, similar to the previous category, its classification terminology expresses the number of processes involved in the conversation. A third category introduces two new patterns to CommDP. This category is called Composition. See Table 7 for a complete list of the patterns currently in CommDP.

Table 7. CommDP Categories and Patterns

Category	Pattern	Acronym
2 participant	Request-Reply	RR
	Request-Reply Acknowledge	RRA
	Idempotent Retry	IPR
	Intermediate State Message	ISM
	Second Channel	2Ch
	Front End	F-E
	Proxy	PXY
3+ participant	Reliable Multicast	RMC
	Publish-Subscribe	P-S
Composition	Sequential Composition	SC
	Nested Composition	NC

The following paragraphs contain descriptions of seven out of the eleven patterns following the language template defined in section 3.5. Every pattern may be composed of other patterns, except, the RR pattern, which becomes part of every peer-to-peer communication, see Figure 11. All other patterns are modeled after this pattern. The reused elements that are lent from other patterns are represented in black in every message sequence representation. The new elements that correspond to the new pattern that is being studied are highlighted on light blue. For example, in Figure 12, the RRA pattern is designed over the RR pattern, the new element is the acknowledge message.

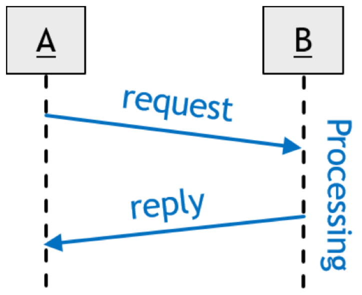


Figure 11. Request-Reply Pattern Structure.

Patterns can be used together to solve some requirements that are not covered by a single pattern. We propose two composite patterns that will help solving this problem. They are Sequential Composite (SC) and Nested Composite (NC). These two patterns are not documented using the template pattern, instead they simply give advice on how two or more patterns can be used together to solve a communication problem. SC intends to use two or more patterns sequentially, i.e. after the final action of pattern A, a second pattern will be implemented in a sequential fashion way, pattern A passes the control of the conversation to pattern B, and this pattern will finish the conversation. A nested composition, on the other side, can include a second pattern in the middle of the conversation between process A and process B, i.e. pattern A gives control to pattern B, then later it takes the control again and finish the conversation, in fact, most patterns ae composed by RR.

IPR and P-S patterns are not covered here, they can be found on their respective authors' publications: Service Design Patterns by Daigneau [51] and Pattern-Oriented Software Architecture Volume 4 by Buschmann et. al. [19]. For their comprehension related to the language we present brief descriptions of them. As other patterns may appear afterwards, they will be defined on the CommDP wiki [53] [http://commdp.serv.usu.edu], and in future work.

3.6.1. Request-reply Pattern

Name and Overview:

The *Request Reply* (RR) pattern is perhaps the most pervasive protocol design pattern; instances or applications of this pattern can be seen in the vast majority of modern communication protocols, including protocols in layers below the application level. It simply involves one process sending a message (a request) to another one and waiting for a reply from that process.

Intent:

RR addresses the problem where a process, A, needs to access or use shared resources in another process, B, with a reasonable degree of reliability and synchronicity. The solution consists of A sending B a message (i.e. a request) and B sending back a message (i.e. a reply) after processing the request. For A, this simple mechanism provides a modest level of reliability and synchronization, because the reply proves that B received the request and can provide relevant information about B’s state. Furthermore, if A does not receive a reply within a specific amount of time (i.e., a timeout), it can resend the request. It can continue to timeout and retry until it eventually receives a reply or it exceeds some maximum number of retries. This “timeout/retry” behavior is the essence of the RR pattern, see Figure 11.

Description:

a) *Problem*: One process A (client) needs to access a shared resource or service provided by another process B (a resource manager), with a modest degree of reliability.

b) *Context*: There is no guarantee of "at most once" semantics needed for the execution of the requested service.

c) *Solution*: The client initiates the conversation by sending a request to resource manager, which receives the request, processes it, and sends by a reply. The client waits for the reply for a certain amount of time, know as the timeout. If it arrives, the conversation finishes successfully. If not, the initiator tries sending the request again. The initiator will eventually give up after a certain number of retries, know as the retry max. This basic behavior is referred to as timeout/retry. A possible variation is that a Request is sent and a Reply is not expected. The following are elements of a design pattern solution:

   - Participants:
      - Initiator
      - Responder.

     In a client-server architecture and in many distributed systems, in general, the client is a conversation initiator and the resource manager is a responder
   - Messages:
      - Request
      - Reply

- Scenario:
    - Initiator -> |Request| -> Responder
    - (Responder processes the Request)
    - Responder -> |Reply| -> Initiator
- Semantic behavior:
    - After the Initiator sends the Request, it waits for the Reply, up to x time units (the timeout)
    - If the reply does arrive with timeout, the Initiator response the Request
    - Attempts are repeated up to y times (the retry max)

Consequences:

Because a message is sent from A to B and a reply is sent back (when B is operating as expected and messages are delivered successfully), the pattern can provide a modest amount of reliability and synchronicity. RR's simplicity allows instances of it to be aggregated into more complex protocols. Also because of its simplicity, it can be easily combined with other patterns.

Known uses:

RR is typically found in: Connectionless communications such as basic HTTP implementations, one way communications, non-reliable communications, asynchronous calls, basic patterns to be reused in other communication patterns, and RPC implementations.

Aliases and Related Work:

This pattern can also be found classified under the following names: Request-Response [29], Request-Reply [29], Single-Request-Response [54].

Examples of Application:

Instances of RR pattern occur frequently in ACP's, including in many standard TCP-based ACP's like Echo [55] [56], SNMP's Agent-Manager communication [57] and HTTP 1.0 [58]. In some cases, a single instance of the RR pattern comprises the entirety for the ACP, as with the Echo protocol. In other cases, many instances of RR are integrated into ACPs as in SNMP. Also they are combined with other types of communications. Following are scenarios for two ACPs that use RR, namely Echo and HTTP1.0:

a) *Echo protocol scenario1*: As a Useful debugging and measurement tool: "An echo service simply sends back to the originating source any data it receives." [56]
b) *Echo protocol scenario2*: An UDP Based Echo Service "Another echo service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 7. When a datagram is received, the data from it, is sent back in an answering datagram."[56].
c) *HTTP1.0 Scenario*: A client establishes a connection with a server to send a request, the client sends the request containing a method, (either GET, HEAD, or POST), the URI, protocol version, client information, and an optional body content. The server responds with a message containing: protocol version, error (or success) code, server information, and perhaps body content [58].

3.6.2. Request-Reply-Acknowledge Pattern

Name and Overview:

The *Request-Reply-Acknowledge* (RRA) pattern, see Figure 12, is a protocol design pattern specially concerned with implementations of "at most once" semantic [59], where the server stores (caches) its reply data in case of a duplicate request, to save processing time for a faster response. Eventually after receiving the acknowledgment, the server disposes of the temporal data, and terminates the conversation. A as much as B are well informed about each other state. At most once semantics, assures that the server executes the request at most once, even when A sends duplicated requests [60]. RRA pattern is usually combined with idempotent retry to allow the "at-least-one" semantics [61] additional to the at-most-once semantic.

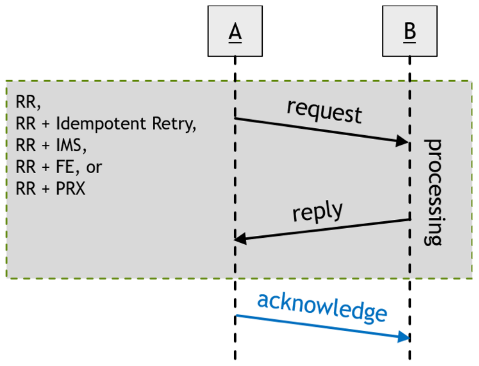


Figure 12. Request-Reply-Acknowledge Pattern Structure.

Intent:

The RRA pattern extends RR solution with a third message (an acknowledgement) that *A* sends to *B* after receiving the reply, and gives *B* a timeout/retry behavior with respect to the sending of the reply and waiting for an acknowledgement. This pattern is useful in situations where significant processing may occur on *A* after receiving the reply or when it is problematic for *B* to reprocess duplicated requests caused by *A*'s timeout/retry behavior. With this pattern, instead of reprocessing duplicate requests, *B* simply caches its replies and resends them to *A* when necessary. The acknowledgement tells *B* that *A* has received the reply and, thus, removes it from its cache. This pattern offers more reliability and synchronization than RR, but at the cost of an additional message.

Description:

a) *Problem*: A client needs to access a shared resource (e.g., a service) provided by resource manager, with a high degree of reliability.
b) *Context*: "at most once" semantics needed for the execution of the requested service, i.e. the responder saves the previous requests, in case of duplicate requests, it replies back without need of processing. The server saves all requests. Once the server receives the acknowledgement, it safely discards that information [59].
c) *Solution*: The client initiates the conversation by sending a request to the resource manager, it receives the request, processes it, and sends by a reply. The client has a timeout/retry behavior while waiting for the reply. When it gets the reply, it sends back an acknowledgement (message has been received) to the Responder. The Responder has timeout/retry behavior while waiting for the acknowledgement. Additionally, the server may cache its reply processed data.

   - Participants:

- Initiator
- Responder.

- Messages:
  - Request
  - Reply
  - Acknowledge (Ack)
- Scenario:
  - Initiator -> |Request| -> Responder
  - (Responder processes the Request)
  - (responder caches the Reply)
  - Responder -> |Reply| -> Initiator
  - Initiator -> |Ack| -> Responder
- Semantic behavior:
  - The Initiator has a timeout/retry behavior on the Reply.
  - The Responder may save the Request to save processing time in case of a duplicate request
  - When the Initiator receives a Reply, it sends back an Ack and considers the conversation successfully completed.
  - The Responder has a timeout/retry behavior on the Ack.
  - When it receives the Ack, it considers the conversation successfully completed and removes the previous request from its cache memory.

Consequences:

One message is sent from A to B with a high level of reliability and synchronicity allowing "at most once semantics" implementations, i.e. B knows that its Reply was received and can remove its temporal (cached) processed reply from its memory.

Known uses:

RRA is typically found in: Reliable communications, secure communications, connection-oriented communications such as the establishment of an HTTPS communication, communications that may need re-transmissions, one way and two way communications, synchronous calls, designing Remote Procedure Calls (RPC) applications, basic pattern to be reused in other communication patterns that need a good level of reliability, stateless servers.

Aliases and Related Work:

Aliases

This pattern can be implemented as Request-Reply with Piggy-Backed Acknowledgements.

Related Work

- Modeling and Formal Verification of Communication Protocols for Remote Procedure Call [62].
- Modeling and Verification of Some Communication Protocols [63].
- Request-Reply-Acknowledge Protocols (section 20.6) [59].
- The Constrained Application Protocol -CoAP (RFC7252 [64]) for improving

REST through TCP performance [65].

Examples of Application:

Instances of RRA occur when the server needs to know that its request was properly received by the client. One typical example is the RPC protocol over UDP [61], where the RCP application must implement its own acknowledgement protocol, which is usually the reply from the client to the server saying that the request was properly received. This is used specifically to implement the "at most once and exactly once semantics", so when the resource manager receives its acknowledge, it discards its cached information. A variation of this protocol can be found on Network File Sharing *write()* operations [61], where an acknowledge can be included in the reply message, to tell the client that the data was received and processed.

a) *RCP over UDP Scenario*: The client sends a request to the server, in the reply, the server can attach the ACK request, then the client sends another ack. A timeout period of 0.5 to 1.0 seconds could be useful in these cases [61].

3.6.3. Idempotent Retry Pattern

The *Idempotent Retry* pattern (IPR) [51] captures a different solution to the problem of processing duplicate requests. Like RR, its solution consists of A sending a request to B with a timeout/retry behavior and B sending a reply back to A. But, unlike RR, the semantics of the protocol dedicate the processing of the request must be idempotent. This pattern applies to situations where the requested processing is relatively light, i.e., less expensive than caching replies. This pattern is completely covered by Robert Daigneau [51] in his book Service Design patterns. An example of this pattern applied is WS-Reliable Messaging, a protocol that defines a standard XML- based vocabulary, this protocol provides message delivery assurances with the capability of try sending messages in case of failed attempts. See Figure 13.

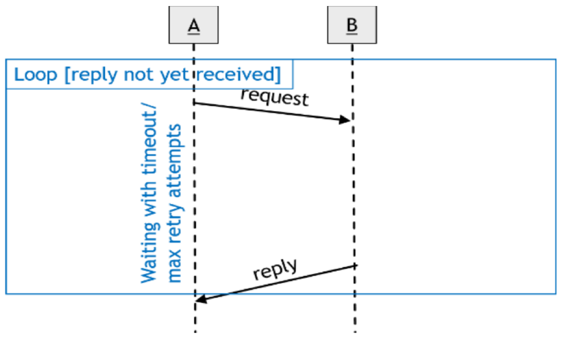


Figure 13. Idempotent Retry Pattern Structure.

3.6.4. Intermediate State Message Pattern

Name and Overview:

The *Intermediate State Message* (ISM) pattern is perhaps the most used pattern for long-running processing requests. Instances of this pattern are found in big data processing, online backup systems, cloud-based storage services. It involves one process sending a request that needs a long-processing time, to another process, and waiting for partial replies ($state_1$, $state_2$, ..., $state_n$) telling something about the state of its request, see Figure 14. The initiator process waits for the final reply from the responder to finish the conversation.

Intent:

ISM, is similar to RR, but addresses the problem of long-running conversations due to request actions taking substantial amounts of time to complete.

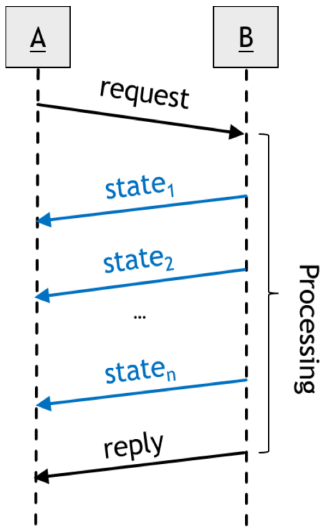


Figure 14. Intermediate State Pattern Structure.

Description:

a) *Problem*: A client needs to access a shared service provided by a resource manager (process B), but the execution of that service is long-running and the client would like to know when key milestones in the processing are reached. And at the end of the communication *B* sends the final message to *A*.

b) *Context*: The processing of the request is long-running with intermediate milestones. "At most once" semantic [59] is needed.

c) *Solution*: After *A* sends a request to *B*, it has *B* send *A* one or more intermediate messages that reflect its current state. For example, *B* may send a message immediately after receiving the request to let *A* know that it got the request, another message when the processing is 10%, another at 20% complete, and so on. Each intermediate message provides state information about *B*, which improves synchronization in the presence of time-consuming actions. The last reply will inform *A* that the process has been completed, and the communication can be terminated. *B* disposes its temporal storage data.

In a variation of ISM, if some data needs to be sent back to *A* a piggy-backed acknowledge [64] variation implementation could be useful. Additional to the state reply, the message could carry back some extra information.

- Participants:
    - Initiator
    - Responder.
- Messages:
    - Request
    - State$_1$
    - State$_2$
    - ...
    - State$_n$
    - Reply
- Scenario:
    - Initiator -> |Request| -> Responder
    - (Responder starts some processing)
    - Responder -> |State$_1$| -> Initiator
    - (Responder do more processing)

    - Responder -> |$State_2$| -> Initiator
    - (Responder do more processing)
    - …
    - Responder -> |$State_n$| -> Initiator
    - (Responder do more processing)
    - Responder -> |Reply| -> Initiator
- Semantic behavior:
    - The Initiator and Responder have similar behavior to the corresponding processes in the RR pattern, except that the initiator needs to receive the state of the processing request and waits (locked by the responder) until it receives the reply. A timeout/retry behavior is implemented based on the n-states and the reply messages.

Consequences:

One message is sent from A to B requesting long processing in B with a high level of Reliability and keeping both processes fully synchronized. The request is processed by *B* and it kept informed *A* about its processing request at all time.

Known uses:

Typical uses are for: Remote File Managers, remote backups, long transaction processes, and remote file transfers in some cases. The latter is more directly related to the Second Chanel pattern (2Ch) though.

Aliases and Related Work:

Related Work

- Long-running transactions [66]
- REST and long-running jobs [67]. Although this is an asynchronous implementation, it is related to long-running processing needs.
- Protocols For Long Running Business Transactions [68].

Although the next three implementations are directly related to multiple responders, they also deal with long-running processing requests.

- Participation Protocol Framework [69]
- WS Coordination Framework, which supports long-running transactions [70].
- Collaboration Protocol Agreement [71]

Examples of Application:

Implementations of this type of behavior are file transfer protocol applications such as FTP (RFC959) [72] or wget [73]. Also Remote Backup Services [74] such as those provided by cloud computing storage and database services. Actually, the ftp protocol is more a complete implementation of a 2Ch pattern, but its implementation of the data connection relies on the ISM pattern. Besides this, anything on a website with a progress bar, which may rely on web sockets is a clear implementation of the ISM pattern.

a) *FTP Data Connection Scenario*: Pre-connection actions

- The user Data Transfer Process (DTP) sends file data to the DTP server or vice-versa.

- The data is sent until EOF is received.

3.6.5. Second Channel Pattern

Name and Overview:

The *Second* Channel pattern (2Ch) is found in applications where there is need of extensive data interchange and the messages can not be used as a carrier for control messages, see Figure 15, i.e. piggy backed acknowledge is not allowed. Instances of this pattern can be found in long running conversations such as the file transfer protocol, where two different ports are used. One for data transfer and one for controlling the transference. Instances of 2Ch can be found in Real Time Communications (RTC) in the browser, for example applications that use the WebRTC [75] framework, where a peer-to-peer communication is open, and a signaling method is used for control messages and to coordinate the communication. Typical implementations are text, audio and video chatting [76]. The File Transfer Protocol (FTP) and its variations are classical examples of this pattern.

Intent:

The 2Ch pattern is for situations involving long-running conversations, but ones dominated by significant amounts of data transfers instead of time-consuming actions. Because large data transfers can delay intermediate state messages, this pattern's solution suggests opening a second communication channel between *A* and *B* that is dedicated to data transfer, leaving the original communication channel available for intermediate state or control messages.

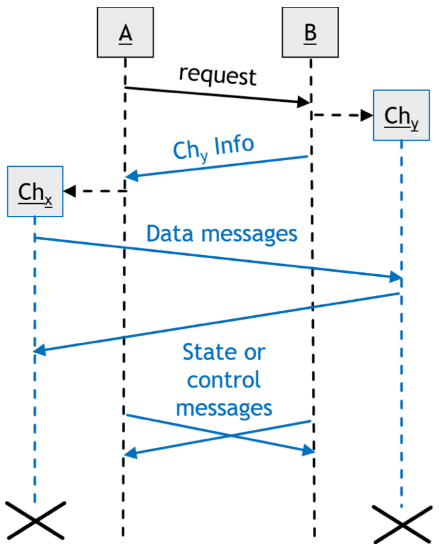


Figure 15. Second Channel Pattern Structure.

Description:

a) *Problem*: One process wants to exchange data with another process, and at the same time, it needs to control the conversation.

b) *Context*: The amount of data is large or the data exchange needs to occur over a long time.

c) *Solution*: The two processes coordinate on the creation of a second, dedicated communication channel for the exchange of sending messages, leaving the first channel open for controlling the conversation.

A possible variation of this pattern would be the case of a third party process handling the control messages.

- Participants:

  - Initiator
  - Responder.
- Messages:
  - Request
  - Ch_y info
  - $State_1$
  - $State_2$
  - ...
  - $State_n$
  - Data $message_1$
  - Data $message_2$
  - ...
  - Data $message_n$
- Scenario:
  - Initiator -> |Request| -> Responder
  - (Responder opens 2nd communication channel)
  - Responder -> |Ch_y info| -> Initiator
  - (Initiator connects to 2nd communication channel (Ch_y))
  - ASYNCHRONOUSLY:
  - {
  - ch_x while(messages)
    - Initiator <-> |$State_1$| <-> Responder

        - Initiator <-> |$State_2$| <-> Responder
        - ...
        - Initiator <-> |$State_n$| <-> Responder
    - ch_y while(messages)
        - Initiator -> |Data messages| -> Responder
        - Responder -> |Data messages| -> Initiator
    - }
- Semantic behavior:
    - The Init Reply (Ch_y info) should contain information necessary for the Initiator to connect to the second communication channel.
    - The Responder's End Point for the 2nd communication channel must be accessible to the Initiator.
    - The individual subsequent replies (Data message interchange) on the 1st communication channel can follow any appropriate communication pattern.
    - The 2nd channel can be bi-directional or uni-directional
    - The data on the 2nd channel can be structured or unstructured
    - Conversation control operations should occur on the 1st channel
    - Conversation is over when, after a timeout, there is no more messages

Consequences:

A conversation between A and B, i.e. a synchronized long running message interchange between *A* or *B*.

Known uses:

2Ch is found in: Long conversations, extensive exchange of messages, real time communications, audio chat, text chat, video chat.

Aliases and Related Work:

Related Work

- Message Channel pattern, as part of Interface partitioning section in Pattern-Oriented Software Architecture Volume 4, (POSA 4), chapter 10 [19]
- WebRTC Framework [75]

Examples of Application:

2Ch pattern occurs in applications such as file transfer protocols, as is the case of ftp [77], which uses a separate control and data connections for communication between the client and the server. A complete description of FTP can be found at RFC959 [77]. Other applications can be peer to peer text, audio and video chat applications, for example, implementations using the WebRTC framework [75].

a) *FTP Scenario*:

- The User-protocol interpreter initiates the control connection,
- Ftp commands containing the action, data port, transfer mode, representation type, and structure are generated and transmitted to the server via the control connection.
- Standard replies are sent from Server Protocol interpreter back and forward.

- The user Data Transfer Process (DTP) listens on the specified port,
- And the server DTP initiates the data connection and data transfer according to the provided commands.

3.6.6. Front End Pattern

Name and Overview:

The *Front End* pattern (F-E) is concerned with decoupling the initiator from the responder. In this sense, the *Front End process* offers an interface to the *initiator*, but, it allows *B* to reply back directly to A, see Figure 16. The localization of the Resource Manager is handled by the Front-End process to provide alternative implementations in case of extreme load or failures of the resource managers.

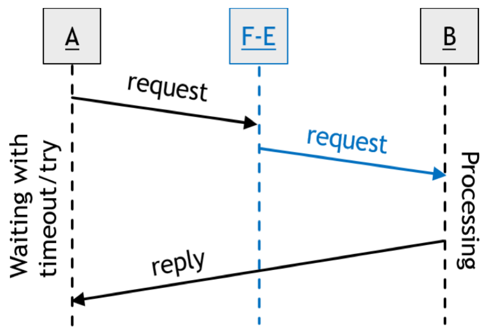


Figure 16. Front End Pattern Structure.

Intent:

F-E addresses the problems of making the location of shared resource transparent to the client, allowing the number of resources to change dynamically. It has a resource client send requests to a front-end process that automatically redistributes them to appropriate resource managers, *B* processes. After processing the request, a resource manager replies back to the client directly. The front-end process can use a variety of criteria to decide how to redistribute requests, including request type, resource type or identity, and resource manager load. By itself, this pattern's primary focus is on the distribution and scalability of resources.

Description:

a) *Problem*: A client needs to access a shared resource or service replicated across one or more resource managers, with a modest degree of reliability.

b) *Context*: There is no need to guarantee "at most once" semantics for the execution of the requested service. The system needs to be scalable or handle spikes in load.

c) *Solution*: The solution is similar to RR, except the request is directed to a front-end process, which then delegates it to one of the resource managers. The resource manager responds back to the requesting process directly.

- Participants:
    - Initiator
    - Front End
    - Responder.

    The client is a conversation initiator and the resource manager is a responder, the Front End is an extra layer of resource manager that forwards the client request and the requestor's end point to the actual resource manager, so the Responder can reply back directly to the Initiator.

- Messages:

    - Request
    - Reply
    - Initiator End Point
- Scenario:
    - Initiator -> |Request| -> Front End
    - Front End -> |Request and Initiator End Point| -> Responder
    - (Responder processes the Request)
    - Responder -> |Reply| -> Initiator
- Semantic behavior:
    - The Initiator and Responder have the same behavior as in the RR pattern
    - The Front-end forwards the Request with the End Point of the Initiator to the Responder
    - The Responder replies directly to the Initiator
    - The End Point has to be accessible to the Responder, so the Front End and Responder should be on the same network
    - The Front-end does not need to track the conversation

Consequences:

One message is sent from A to B with a certain level of reliability and synchronicity through a third process (Front End) that handles A's request. B sends back a reply message to A hiding B’s details. This pattern provides a high level of scalability.

Known uses:

F-E pattern is found usually in: Load balancers and authorization services such as a simple login.

Aliases and Related Work:

Aliases

the requestor [78]

Related Work:

The authenticator pattern [79], Media Type Negotiation [80] [51]; requestor pattern, as part of the Distribution Infrastructure section in Pattern-Oriented Software Architecture Volume 4, (POSA 4), chapter 10 [19].

Examples of Application:

Occurrences of this pattern can be found in implementations of load balancers as is the case of NGINX. NGINX is an open-source light weight web server or proxy server. It loads balance requests across multiple application instances for optimizing resource utilization, maximizing throughput, reducing latency and ensuring fault-tolerant configurations [81] [82].

NGNIX contains configurations for multiple application instances. The configurations for these application instances can be static or can be created on the fly. It can use a variety of load balancing methods such as round-robin, ip-hash, least-connected server, generic hash, or server with lowest average latency. NGNIX as a front-end can support many other rich features besides just load balancing, such as re-routing the requests according to their supported sessions, authorization(s), health monitoring of the running servers, and identifying malicious requests.

a) NGINX Scenario:

- Any web-based client request first goes to the NGNIX load balancer
- Based on the request header “Host”, NGINX decides which of the listed server(s) will process the request
- In case of multiple hosts, it can apply one of the load-balancing methods described above.
- If NGINX could not find any better information, it will re-route the default server
- Once a relevant server processes the request, this server sends back the reply to the client and does not re-route to NGINX first.

3.6.7. Proxy Pattern

Name and Overview:

The *Proxy* pattern (PXY) is perhaps the most used to decouple a conversation between a client and its resource manager(s), see Figure 17. Similar to F-E, this pattern offers location transparency and authentication, data format conversions and protocols compatibility. Instances of this pattern can be seen in multi-platform architecture applications such as CORBA [83], especially in its Object Request Broker ORB implementations, in SOAP-REST-SOAP API conversion services, in web services that provide different data formats such as: XML, JSon, Plain text, SQL.

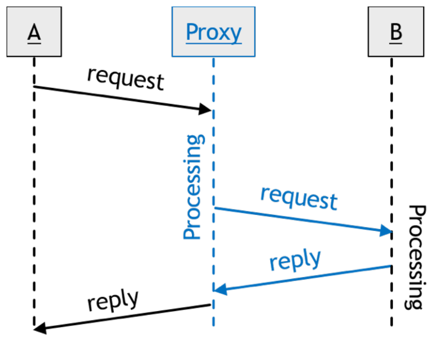


Figure 17. Proxy Pattern Structure.

Intent:

Like F-E, PXY introduces a process between a resource client and a resource manager. However, the intermediate process, called a proxy, serves other functional purposes besides re-distribution of the requests, for example it may provide authentication, access control, audit logging, and data transformation functionality. Also, the resource manager returns replies through the proxy to client, completely isolating the client from the resource manager.

Description:

a) *Problem*: A client needs to access a shared resource or service provided by one or more resource managers, but the resource managers are not directly accessible to clients.

b) *Context*: No “at most once” semantics needed. The resource manager needs to be protected against direct access or attacks. The system needs to be scalable or handle spikes in load.

c) *Solution*: The solution is similar to the RR with a Front-end pattern, except that the Front-end is replaced with a proxy process and replies are returned through a proxy instead of directly to the initiator.

- Participants:
    - Initiator
    - Proxy
    - Responder

    The client is a conversation initiator and the resource manager is a responder, the Proxy is an extra layer that forwards the client request to the resource manager and also forwards the reply from the server to the client.
- Messages:
    - Request
    - Reply
- Scenario:
    - Initiator -> |Request| -> Proxy
    - (Proxy processes |Request|)
    - Proxy -> |Request| -> Responder
    - (Responder processes |Request|)
    - Responder -> |Reply| -> Proxy
    - (Proxy processes |Reply|)
    - Proxy -> |Reply| -> Initiator
- Semantic behavior:

- The Initiator and Responder have the same behavior as in the RR pattern
- The Proxy forwards messages from the Initiator like a Front-end, but no End point is needed.
- The Proxy has to keep track of each conversation in progress so it can forward a Reply to the correct Initiator and in the correct conversation context, i.e. Proxy must handle Initiator and Responder's end points.

Consequences:

One message is sent from A to B with a certain level of reliability and synchronicity through a third process (proxy) that handles A's request. B replies back to the proxy and this sends the reply to A. This pattern provides a high level of scalability, also it protects process B from attacks, by hiding its location, i.e. location transparency.

Known uses:

Login services, security, object brokers, middleware design, and resources protection. In general, applications that need access or location transparency.

Aliases and Related Work:

Aliases

Client Proxy [19]

Related Work

Service controller [28] [51], message translator [19], client request handler pattern as part of the distribution infrastructure section in Pattern-Oriented Software Architecture Volume 4’s chapter 10 [19], client proxy pattern as part of distribution infrastructure section in POSA 4's chapter 10, proxy pattern as part of interface partitioning section in POSA 4's chapter 10, broker pattern as part of interface partitioning section in POSA 4's chapter 10.

Examples of Application:

HTTP Proxy, DNS (cached responses and reply back), VPN are some protocols implemented using the PXY pattern.

3.6.8. Reliable Multicast Pattern

Name and Overview:

The *Reliable Multicast* pattern (RMC) is widely used in multicast or broadcast communication systems, where the request of processing is delegated to *n* Resource managers (RM), where $n>1$, and the execution needs to be synchronized in the *n* RMs. It may need a simultaneous processing of the request in every responder. Instances of this pattern can be found in bank transactions, parallel computing, etc.

Intent:

The RMC pattern, Figure 18, builds on the multicast idiom to provide reliability and synchronization among a group of processes. Its solution is a protocol that starts with a process *A* sending a request message to a group of process, $B = \{ b_1, .., b_n \}$. Each process b_i sends a reply back to *A* when it receives the request and is ready to process it. After *A* receives reply from all *B* processes, then *A* will multicast a go-ahead message back out to all *B,* this message indicates that they can proceed with the processing of the request. In this way, the execution of the request is synchronized among all of the *B* processes. If *A* fails to receive a reply from every *B* process, it can resend the request to some or all of them until it gets a reply from all of them or terminates the conversation as failed. This pattern focuses on providing strong reliability and synchronization, but can also help with scalable distribution of resources.

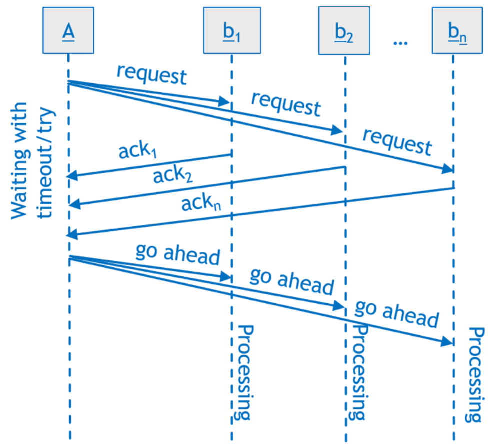


Figure 18. Reliable Multicast Pattern Structure.

Description:

a) *Problem*: A client needs to access resources provided by one or more resource managers. And the resource managers have to process the request all at-once, in a transaction fashion-way.

b) *Context*: There is need of synchronized execution of a set of requests sent by only one initiator and several responders.

c) *Solution*: The client initiates the conversation by sending *n* requests to *n* resource managers, which receive the request, but do not process it immediately. Every resource manager sends back an acknowledgement telling the initiator that they received the request and are ready to process it. The Initiator will send a "go-ahead" message to every RM after it has received the *n* acknowledge messages. RMs process all the requests.

One variation of RMC would be its combination with RRA after processing the request in every RM.

- Participants:
  - Initiator
  - Responder_1.
  - Responder_2.
  - …
  - Responder_n.

  The client is a conversation initiator and there is a set of resource managers that behaves as responders, every responder is responsible for sending an Ack, and the client controls the execution of the requests in the responders.
- Messages:
  - Request
  - Ack_1
  - Ack_2
  - …
  - Ack_n
  - Go ahead

- Scenario:
    - (A gets a list of Responders)
    - Initiator -> |Request| -> {Responder$_1$, Responder$_2$, ..., Responder$_n$}
    - Responder$_1$ -> |Ack$_1$| -> Initiator
    - Responder$_2$ -> |Ack$_2$| -> Initiator
    - ...
    - Responder$_n$ -> |Ack$_n$| -> Initiator
    - Initiator -> |go ahead| -> {Responder$_1$, Responder$_2$, ..., Responder$_n$}
    - Synchronously
        - (Responder$_1$ processes |Request|)
        - (Responder$_2$ processes |Request|)
        - ...
        - (Responder$_n$ processes |Request|)
- Semantic behavior:
    - After the Initiator sends the Request to n-Responders, it waits for the acknowledgement of every Responder, up to x time units (the timeout).
    - Responders wait for "go ahead" message from Initiator before start processing the Request.
    - The Initiator has a timeout/retry behavior on the Acks.
    - When it receives the n-Acks, it sends the "go ahead" signal to the n-Responders
    - Initiator considers the conversation successfully completed

Consequences:

A sends a request to multiple responders, this request is processed in all the responders synchronously with a good level of reliability.

Known uses:

RMC pattern can be found in many broadcasting and multicasting applications such as: Multicast communication, conference groups, news feeds, message distribution, wireless communications, digital-TV for content protection / paying users, ads, software updates, software deployment.

Aliases and Related Work:

Multicast protocol [84]

Examples of Application:

A well-known application of this pattern is JGroup, JGroup communication uses the term group and member. A member is a node, i.e. a process residing on one or different hosts, and a group is a cluster, which can have one or more nodes belonging to it [85, p. 2]. The main component of JGroup is the channel, it is simple and primitive and provides asynchronous message sending and reception, similar to UDP. It is connected to a protocol stack. Whenever the application sends a message, the channel passes it on to the protocol stack, which ultimately gets pushed on the network. Similarly, the protocol stack receives the message and ultimately pushes it on to the channel [86].

a) *JGroup scenario*: In this reliable group communication, processes can join a group, send messages to all members or single members and receive messages from members in the group. A group is identified by its name. When a process joins a non-existing group, the group will be created automatically. The system keeps track of the members in every group and notifies group members when a new member joins, or an existing member leaves or crashes.

3.6.9. Publish-Subscribe Pattern

Finally, the *Publish-Subscribe* [19] pattern (P-S) is a powerful mechanism for decoupling message senders (publisher) from message receivers (subscribers) see Figure 19. With this pattern, an intermediate process acts as a store-and-forward buffer for message transmission with the capabilities for managing subscribers and delivering individual message to multiple subscribers. We refer to this pattern as explained in [19].

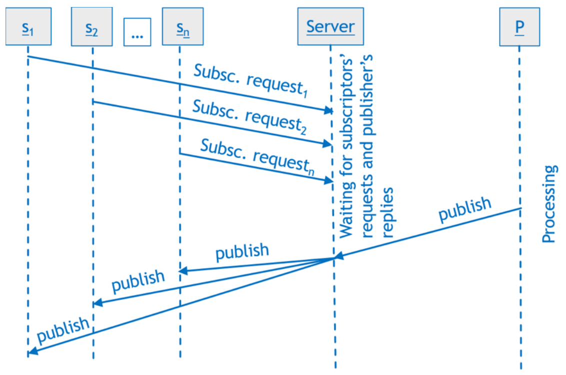


Figure 19. Publish-Subscribe Pattern Structure.

3.6.11. Quality Analysis of CommDP Patterns

Developers can select the appropriate pattern according to the application needs and its consequences based on the RSLA qualities. Following are the analyses of each

RSLA quality based on their attributes. This arbitrary scale was explained in section 3.4.

Table 8. Analysis of Reliability for CommDP's Patterns

Category	Pattern	Reliability attributes			
		Failure Detection	**Failure Discrimination**	**Failure Handling**	**Handling of Duplicate Messages**
2 participant	RR	1	1	1	1
	RRA	2	1	1	1
	IPR				2
	ISM	1	1		
	2Ch				
	F-E				
	PXY		1	1	1
3+ participant	RMC	2	1	1	1
	P-S				1
Composition	SC				
	NC				

Table 9. Analysis of Synchronicity of CommDP's Patterns

Category	Pattern	Synchronicity attributes	
		Coordination	**Message Ordering**
2 participant	RR	1	
	RRA	2	
	IPR		
	ISM	2	
	2Ch		
	F-E		
	PXY		
3+ participant	RMC	2	
	P-S		
Composition	SC		
	NC		

Table 10. Analysis of Adaptability for Scalability of CommDP's Patterns

Category	**Pattern**	**Adaptability for Scalability**		
		Location Transparency	**Load Awareness**	**Load Balance / Scalability**
2 participant	RR			
	RRA			
	IPR			
	ISM			
	2Ch			
	F-E	2	1	2
	PXY	2	2	2
3+ participant	RMC	1		
	P-S	2		
Composition	SC	Determined by subpart patterns	Determined by subpart patterns	Determined by subpart patterns
	NC	Determined by subpart patterns	Determined by subpart patterns	2

Table 11. Analysis of Longevity of CommDP's Patterns

Category	**Pattern**	**Longevity attributes**
		Long Running Operations
2 participant	RR	
	RRA	
	IPR	
	ISM	1
	2Ch	2
	F-E	
	PXY	
3+ participant	RMC	
	P-S	
Composition	SC	
	NC	

Finally, when we need to select a specific pattern for a protocol, we can, first, determine a binary selection, where the protocol is analyzed based on the 4 RSLA qualities, see Table 12. Then we match this requirement with the sub-qualities, and we get one or more choices of patterns. If we get more than one optional patterns to get the best option, we analyze which is the pattern that possesses the highest ranking values.

Table 12. Application Examples to Select the Appropriate Pattern

Domain	**R**	**S**	**L**	**A**	**Advised Pattern (s)**
Files Transfer	X		X		Intermediate State
Chatting		X	X	X	Second Channel + Front End
Medical	X	X		X	Reliable Multicast + Second Channel
Transportation	X	X	X	X	Intermediate State Message + proxy
Security (log on, log off)	X			X	Publish-Subscribe
news	X			X	Reliable Multicast NS/OR Publish-Subscribe
Mail/messages	X		X		Intermediate State Message

For example if we need to implement a protocol for file transfer, that supports reliability and longevity, and is not concerned with Synchronicity and Adaptability, possible options, according to the reliability sub-qualities, would be RR, RRA, ISM, PXY, and RMC, see Table 8. We also need that the protocol supports Longevity, hence according to Table 11, there are two possible patterns, ISM and 2Ch. Evaluating both of the RSLA qualities, it is clear that the chosen pattern is ISM. Same analysis can be done for the other protocols shown in Table 12.

3.6.10. Pattern Composition

Patterns are rarely used in isolation; instead, developers combine their solutions to solve complex problems. Virtually any of the CommDP patterns could be combined with any other pattern, but the more useful combinations are ones that have complimentary characteristics, like Request-Reply with Second Data Channel or Request-Reply

Acknowledge with Front End.

To ensure that the CommDP pattern set was as minimal as possible, we did not include in any pattern in CommDP that was simply an aggregation of two or more patterns. For example, there is a common type of distributed system that deals with information flow and processing. In such systems, a process A might send a request to B through a series of intermediate proxy-like processes that transform or augment data in request on its way to B. At each intermediate step, a reply is sent back to A, informing it of the message's process. Eventually, when the transformed message arrives at B and processes it, then B sends a final reply message back to A. This particular solution offers good reliability, synchronization, longevity, and adaptability to scalable distribution, but it is actually just a composition of PXY (applied perhaps multiple times) and ISM.

3.7. Summary and Future Work

CommDP quality model definition led us to conclude that we need to add patterns to cover the attributes that the current patterns in CommDP are not able to implement. For example, *go-back-n* and *selective reject* patterns would implement more reliable protocols. *Message-numbers* and *hold-back queue* patterns would implement more reliable and synchronized conversations. *Load information* and *Timing information* patterns would help on improving long running conversations and reliability.

Other qualities as much as other patterns can be added by collaboration with other professionals to improve the wiki, hence CommDP pattern language, its template, its qualities, and its applicability can be extended to cover more fields on the communication protocols design.

To our knowledge, no one has formally identified similar desirable qualities for communication software protocols, this set of attributes intends to initiate a discussion on the need of a specific quality model that deals with communication software, and also a validation of this model is needed in a future research.

Since every networking application needs some level or synchronicity, every pattern provides some synchronicity, i.e. the sender process blocks after sending its request. And it will increase its needs accordingly, for example if the two processes depend on each other execution for further processing, both processes can be mutually locked each other.

CommDP pulls together reusable solutions to reoccurring design problems with ACPs, filling a much needed gap in the knowledge base for developers of distributed systems. We have characterized the nature of the problems that the CommDP patterns address and the consequences of their solution in terms of four desirable qualities: reliability, synchronicity, longevity, and adaptability for scalable distribution. These qualities are both instructive and discriminating, in that they can help a developer understand the solutions and choose the most appropriate solution for a given situation (problem + context). However, more work needs to be done to formalize these qualities and to solidify the sufficiency and completeness relative to communication-protocol design.

We intend in future research to use CommDP to improve the learning process in students taking courses related to distributed systems. Studying its usefulness in designing and developing applications and its effectiveness in the learning and thinking of students could be valuable for the education and enterprise communities.

CHAPTER 4

USING CLOUD SERVICES TO IMPROVE SOFTWARE ENGINEERING EDUCATION FOR DISTRIBUTED APPLICATION DEVELOPMENT[7]

4.1. Abstract

Software-engineering education should help students improve other development skills besides design and coding. These skills, referred to here as A2R (Analysis to Reuse), include analysis, technology evaluation, prototyping, testing, and reuse. The need for improved A2R skills is particularly pronounced in advanced areas like distributed application development. Hands-on programming assignments can be an important means for improving A2R skills, but only if they focus on the right details. This paper presents a case study of programming assignments offered in a graduate-level class on distributed application development, where the assignments required the students to use cloud services and programming tools that were heretofore unfamiliar to the students. Direct observation by the instructor and a post-class survey provided evidence that the assignments did in fact help students improve their A2R skills. The post-class survey also yielded some interesting insights about the potential impact of well-designed programming assignments, which in term led to ideas for future research.

4.2. Introduction

Imagine yourself at a worktable with four or five of your peers. In the center of the table is a pile of seemingly random objects, including two dozen sheets of paper,

[7] Jorge Edison Lascano, Stephen W. Clyde

paper clips, a small roll of tape, pins, and several small wooden sticks. A quick glance around the room reveals a dozen other groups just like yours with similar piles in front of them. An individual, who is introduced as your customer, stands at the front of the room and says that you have 30 minutes to build a “great” tower. What do you do first? How do you put all that you know about paper, clips, tape, wooden sticks, etc. into practice to satisfy the customer’s request for a tower and do so within 30 minutes?

Such is the typical scene on the first day of class in the undergraduate introductory course on software engineering at Utah State University (USU). In general, all the students have a good working knowledge of objects at their disposal and even some inkling on how they may combine several of them to create new more structural useful objects. Most groups succeed in creating something that stands on its own and roughly resembles a tower within 30 minutes. However, at the end of that time, the customer surprises the students by giving them a few more objects, e.g., more paper and tape, and asks them to take 15 more minutes to make their towers taller or stronger. Many groups fail to do so in the limited allotted time. In fact, about half of them end up destroying their original towers in the attempt.

Afterwards the instructors and students discuss the experience in terms of what worked well for the group, particular difficulties that hindered progress, how the group organized itself, and how they decided on an overall approach. The discussion usually leads to some very interesting comparisons with common aspects of software engineering, such as group work, tool evaluation, prototyping, design patterns, testing, extensibility, reuse, and more. Over the years, one of the authors, who is a long-time instructor for this introductory software engineering course, has observed the following:

1. Virtually no student or group ever asks the customer what a "great" tower means. Most assume that they already know and proceed to build without each researching the requirements.
2. Virtually no student or group ever looks around to see what other groups have done or are doing, evaluates the ideas they see, and then tries to adapt or improve on them.
3. Only a small percentage of the groups try prototyping an idea to explore its characteristics.
4. Only rarely does a group test the properties (e.g., stability or strength) of a component or the whole tower and then try to make modifications to improve those properties.
5. Only a few groups try to establish patterns or "best practices" either in their building processes or the components they create, and then reuse those ideas.

Each of these observations represents a potential engineering pitfall or negative practice that can lead to inefficiency or failure. Software-engineering education needs to help students avoid these and other related pitfalls by connecting theory with best practices in the context of real non-trivial problems [124]. Doing so goes well beyond teaching the "How To's" of a specific technology, like a programming environment. Instead, it requires educators and students alike to address the "How To's" of the overall development process, including:

1. How do we know when we understand the customer's problem sufficiently?

2. How can we benefit from existing technology or from what others have tried in the past?
3. How can we prototype part of a problem or alternative solutions to answer critical questions?
4. How can we test what we build?
5. How can we find good solutions to reoccurring problems and reuse that knowledge?

More concisely, software-engineering education needs to help students make analysis, technology evaluation, prototyping, testing, and reuse an effective and integral part of their development activities [124]. Here, we'll refer to these as Analysis to Reuse (A2R) skills.

The need for better A2R skills is prevalent in every software-engineering domain, but is pronounced in the development of distributed applications. Distributed-application development, or distributed-system development at large, has all of the challenges of traditional software development, plus the complexities introduced by inter-process communications, concurrency, the potential of partial failure, and replication that exist for performance improvements or fault tolerance [69].

Now let us roll our classroom scene forward several years to a graduate software-engineering class that focuses on distributed applications. Students entering in this class have solid foundations in software-engineering fundamentals, programming languages, inter-process communications using sockets, and many other areas of computer science. Yet, they still need to strengthen their A2R skills, especially in the context of distributed applications, and the best way to do that is through hands-on experience [35]. So, from an education perspective, the challenge is to provide realistic and engaging assignments that will strengthen the A2R skills and are doable within the allotted time.

Because distributed applications are relatively complex [125] by their very nature, there are two negative tendencies for program assignments in this area: a) abstracting away too many interesting aspects of the problem and b) getting bogged down with unnecessary application-domain details.

The first tendency is very common in advanced CS courses, because simpler assignments are more manageable, teachable, and easier to fit within a given allotted time. Advanced courses usually have to operate within same time constraints as introductory courses. Even though, they are more complex, it is essential that advanced assignments include reasonable limits on the expected time and effort [125]. Simplicity in their design is a necessity and by itself is not a problem. Focusing on the wrong details and abstracting away all interesting parts of the problem, however, is a serious real pitfall. For example, scalability is a real and very common aspect of most distributed applications [69]. Even though removing scalability requirements would simplify an assignment, it would rob the students of a valuable opportunity to improve A2R skills in a relevant area.

The second tendency is to allow an assignment to get bogged down in application-domain details, shifting focus away from the learning objectives. Assignments in advanced courses, like distributed-application development, work best if they are grounded in a meaningful real-world domain. However, most distributed applications and their domains are relatively complex. If not careful, an instructor could easily use all available time explaining the sample application domain, instead about the core course topics. Keeping assignments focused on a small set of functional requirements that require minimal application-domain knowledge, is essential to making sure that they are doable within time limits and achieve the learning objectives.

This paper describes a case study of programming assignments conducted in an advanced software-engineering class on distributed-application development, where all of the assignments required students to use cloud resources for their execution environment. The hoped-for result was that the assignments provided students significant opportunities to improve their A2R skills, while introducing them to new concepts and development tools. Section 2 describes the course's programming assignments in terms of their learning objectives, the application domains that act as backdrops, and their requirements. Section 2 also explains the tools and technologies introduced for each assignment. Section 3 summarizes the instructor's observations made throughout the semester and assignment design learnings. To evaluate the effectiveness of the assignments, we conducted a post-class survey. Section 4 describes this survey and presents the resulting raw data. Since the class was a second-year graduate class, the enrollment was small. So, we cannot make many generalizations from the survey data. Nevertheless, they do lead us to some interesting insights. We share those insights in Section 4.B. Section 5 explores related work in software-engineering education using cloud resources and hands-on learning. Finally, Section 6 provides conclusions, along with ideas for future research that could further advance software-engineering education relative to A2R skill development.

4.3. Programming Assignments for a Distributed Application Development Course

CS 6200 at USU is a second-year graduate course in software engineering that focuses on the development of distributed applications. Its prerequisite, CS 5200, provides students with a strong foundation in inter-process communication, protocols, concurrency, and communication subsystems. CS 5200 is also a programming intensive course, which means that students who successfully complete it have confidence in their ability to implement non-trivial software systems. The overall learning objectives for CS 6200 are as follows:

- Master theoretical elements of distributed computing, including: models of computation and state, logical time, vector timestamps, concurrency controls, and deadlock;
- Become familiar with the provisioning and use of virtual computational and storage resources in a cloud environment;
- Become familiar with cloud-based tools for processing large amounts of data efficiently; and
- Become familiar with distributed transactions and resource replication.
- For the Spring-2015 semester, the homework was broken down into five assignments, each lasting two to three weeks.

4.3.1. Assignments 1 & 2 – Disease Tracking System

For the first two assignments, the student implemented a set of processes that worked together to form a disease tracking and outbreak monitoring system. They had to deploy multiple processes on EC2 instances within Amazon Web Services (AWS) cloud.

The first type of processes were simulations of Electronic Medical Records Systems (EMR's) that randomly generated notifications of diagnoses for infectious diseases, like influenza. The EMR's sent these disease notifications to Health District Systems (HDS's), which collated diagnoses and then sent periodic disease counts to Disease Outbreak Analyzers (DOA's). Each DOA monitored outbreaks for a single type of disease. See Figure 20. Programs Built as Part of Assignments 1 & 2, Plus an Illustration of Sample Processes. The specific learning objectives for these two assignments included:

- Review inter-process communications;
- Become familiar with vector timestamps and how they behave in a distributed system under varied conditions;
- Become familiar with setting up and using computational resources in a cloud, e.g., AWS; and
- Become familiar with setting up a simple name service.

The students were asked to learn and use Node.js as the primary programming framework [27] [126]. Because Node.js was new to all the students, some class time was dedicated to teaching Node.js, but only enough to get them started. Their unfamiliarity with Node.js was also the reason this first system was split into two assignments. They built and tested approximately half of the functionality in the first assignment and the remainder in the second.

To deploy their systems to EC2 instances on AWS, the students had to learn about security on AWS, create security keys, and setup their own user accounts using Amazon's

Identity and Access Management (IAM). They also had to setup and learn the AWS's command-line language interface (AWSCLI), so they could automate the deployment and launching of their systems.

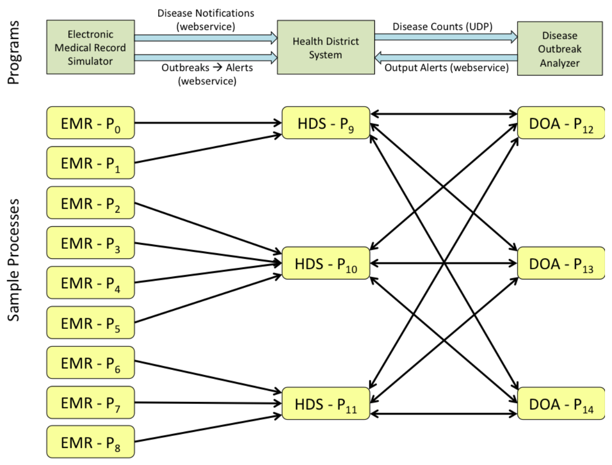


Figure 20. Programs Built as Part of Assignments 1 & 2,
Plus an Illustration of Sample Processes.

4.3.2. Assignment 3 – Twitter Feed Analysis

In this assignment, the students explored how to process big data using MapReduce on AWS and how to configure cloud resources using AWS's Cloud Formation tools. Specifically, they were to capture tweets through Twitter's API and then analyze them for positive or negative sentiment relative to some key phrase, like "health care". The learning objectives for this assignment were as follows:

- Become familiar with setting up and using MapReduce with a cloud-based distributed file system;
- Become familiar with tools for provisioning collections of resources that are needed for a distributed system; and

- Explore the types of problems that are well suited for a MapReduce solution

To complete this assignment, students setup and learned how to use AWS's S3, MapReduce, and Cloud Formation services. Some also used this assignment to learn about a Node.js module for working directly with AWS; while others strengthened their knowledge of AWSCLI.

4.3.3. Assignment 4 – Distributed Election

In this assignment, the students implemented a distributed system consisting of dozens of processes that shared access to common data files, which were collectively treated as one large shared resource, like a database. One of the processes played the role of Resource Manager (RM) and accessed the common data files in response to requests from the other processes. If RM died, then the other processes had to detect that failure and elect one of the remaining processes to be the new RM seamlessly. The learning objectives for this assignment were:

- Master at least one distributed election algorithm;
- Master the concept of resource managers for controlling access to share resources; and
- Become more familiar with tools for provisioning collections of resources in a cloud.

To complete this assignment, we allowed students to use any of the tools they had learned thus far, but they had to deploy their systems to multiple EC2 instances and demonstrate that the system would elect a new RM if the current one was stopped. They had to show that the system has as a whole, lost no work.

4.3.4. Assignment 5 – Distributed Transactions

In this assignment, the students had to build a simple transaction management system with locking capabilities. Like Assignment 4, this system had to support multiple concurrent worker processes, but went a step further in requiring multiple shared resources and multiple concurrent RM's. Each RM had to keep track of a single resource and support lock, read, write, and unlock operations on that resource. The system also had to include a transaction manager that supported starting, committing, and aborting of transactions. Assignment 5's learning objectives included:

- Become familiar with locking; and
- Become familiar with transaction management in a distributed system.

Like Assignment 4, the students could use any of the tools that they learned to this point in completing Assignment 5.

4.4. Instructor Observations

Seven students took CS 6200 in the Spring-2015 semester: 5 who were registered for credit and 2 who audited the class. It is impossible to recap all that took place during the semester, but we summarize a few observations prior to presenting the post-class survey to help set the stage for the survey and our conclusions.

First, we observed that all of seven students started the class with roughly equivalent software-engineering backgrounds and programming skills, even though they were not all seeking the same degree nor did they have the same programs of study. None of the students had used Node.js before and only one had any exposure to cloud computing, and that was only a light exposure.

Second, we observed that requiring students to setup and managing their own cloud resources not only helped them with core concepts and development skills, but it also allowed them to improve their A2R skills relative to figuring out what the most important requirements were, tool evaluation, and testing. For example, in the first two assignments, the students had to deploy their system to EC2 instances. For most of the students, this was the first time deploying something that they built to an execution environment different from their development environment, along an execution environment consisting of multiple virtual machines. It opened their eyes to new challenges, such as firewall issues, file permissions, and missing dependencies. Time was made available in every class period for them to talk about the challenges that they were facing and get ideas from other students or the instructor about how to address those challenges. Similar discussion also took place on an online forum. By the end of Assignment 2, the classroom and online discussions showed that the students had stepped up their efforts to understand the assignment requirements, evaluate the tools available to them, and test their work.

Even though the purposes of Assignments 4 and 5 were considerably different from the first three, they possessed some of the same challenges, like resource name resolution and deployment into a cloud environment. It was encouraging to see that the students solved these problems by adapting techniques used in the earlier assignments and improving upon them – evidence of them practicing A2R skills.

We were happy to see that the students learned some unexpected, but very relevant lessons. For example, one student stored his access keys in a text file and committed that file to a public Git repository. It was not long before someone hacked his AWS account. Amazon and the student caught the problem relatively quickly and simultaneously, but not before the hacker had used over $600 of resources. He ended up taking extra time learning more about security from unauthorized use. Thankful, Amazon worked with him to recover the expenses, so he did not have to pay for the lost out of pocket. Still, it proved to be a valuable learning experience that he will not forget.

With respect to the selected cloud AWS, we observed that it provided a mature and full-featured set of services for the students to learn from. In some areas, AWS's learning curve was steeper than necessary, but with supplementary examples and good discussions, it was manageable. From an education perspective, a good thing about AWS is that it has features in three main categories: Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS) [69].

One negative experience with AWS occurred during Assignment 3, which depended on an Amazon-provided template for setting up a MapReduce cluster. That template was changed by its authors in the middle of assignment, causing several of the students not to complete all of the requirements. To avoid this problem in the future, the instructors will make private copies of public or external resources, so changes to them will not affect assignments in progress.

A. Assignment Design Learning

When instructors design assignments oriented to networking or distributed applications, they need to consider distribution concepts, but at the same time bear in mind the limitations for the students' capabilities and hardware environment. Before cloud resources became available, this typically consisted of one computer [127] or small number of computers on a local area network (LAN) in a school lab. Assignments that work well on one computer or a LAN may not allow the students to gain appreciation for more realistic networking challenges, performance issues, and reliability problems [31]. With cloud resources, assignments can now be designed having a broad range of resources in mind, while still considering good software-engineering practices for analysis, technology evaluation, testing, deployment and even reuse.

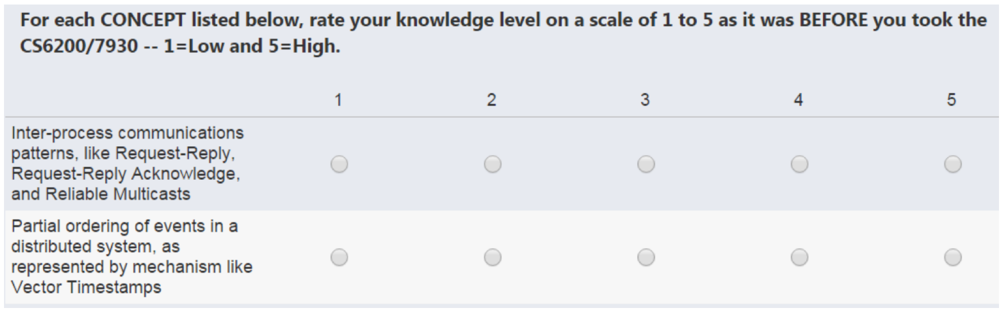
4.5. Post-class Survey

To assess the value of the programming assignments for CS 6200, we designed a post-class survey and conducted that survey with two populations: students registered in CS 6200 for credit and students who just audited the class. Those registered for credit had to complete all of the assignments to receive a grade; those just auditing the class did not. In fact, it is important to note that none of the second group completed any assignment.

4.5.1. Survey Design

We organized the survey into two parts. The first part asked students to rate their knowledge and skills in areas related to the course and the assignments, as they were before the class started, using a 1-to-5 scale. The second part asked them to do the same relative to the end of class.

Table 13 and Table 14 list the concepts (knowledge areas) and skills respectively covered in both parts of the survey. The survey used a Matrix Table format, with the concepts and skills as rows and possible ratings as columns. See Figure 21 for partial view of the survey instrument for Part 1.



For each CONCEPT listed below, rate your knowledge level on a scale of 1 to 5 as it was BEFORE you took the CS6200/7930 -- 1=Low and 5=High.

	1	2	3	4	5
Inter-process communications patterns, like Request-Reply, Request-Reply Acknowledge, and Reliable Multicasts	○	○	○	○	○
Partial ordering of events in a distributed system, as represented by mechanism like Vector Timestamps	○	○	○	○	○

Figure 21. Partial View of the Survey

The difference between each individual's answers to corresponding questions from two parts provides a glimpse of that person's perceived change in knowledge or skill levels as a consequence of the course.

We could have administered a pre-class survey similar to the first part, but considerable differences in each student's personal rating scheme would likely have evolved over the semester, making it difficult to assess perceived change. We could have also administered pre and post exams to measure their proficiency objectively, but there was no common knowledge basis for a pre exam. So, the study would have degenerated into the interpretation of just post exam results.

4.5.2. Survey Results

All seven students completed both parts of the survey. Figures 22 and 23 show averages of the students' raw estimates of their knowledge and skill levels for before and after class. The blue lines represent the levels before and the red lines after. The (a) graphs are for the first population, namely the students who registered for credit and the (b) graphs are for the auditing students. Figure 24 shows the average net change in the levels, broken down by the two populations.

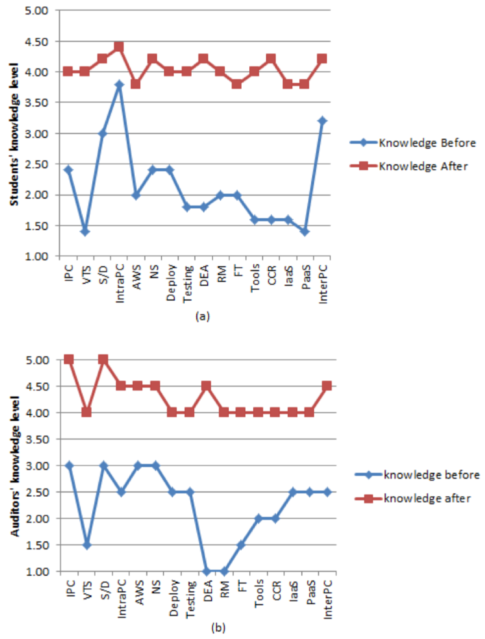


Figure 22. Before and After Knowledge Levels.

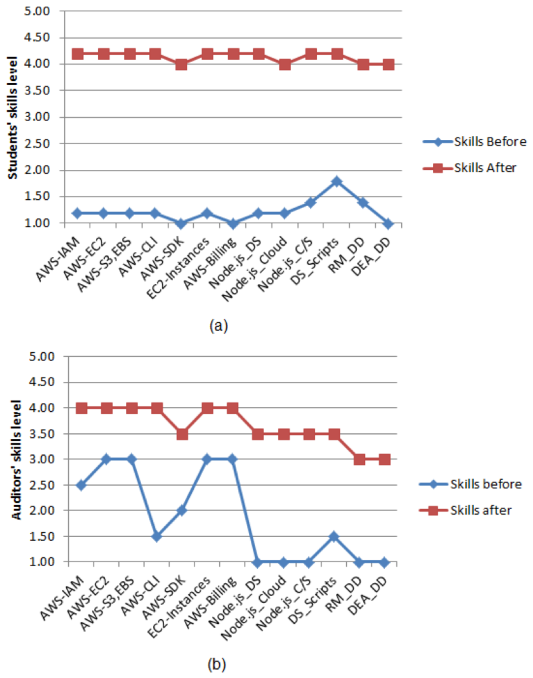


Figure 23. Before and After Skills Levels.

One interesting result that is worth pointing out immediately, is that the first group of students, in general, rated their before-classes level lower than the second population. We believe that this can be contributed to the common adage, “You do not know what you do not know”. The first group of students did the assignments and soon discovered how much they really did not know, whereas the second group did not come to the same realization. For example, the auditors’ perception about their AWS and Node.js skills was that they knew those technologies relatively well before starting the class; meanwhile the first group of students came to realize that their skills were almost nil.

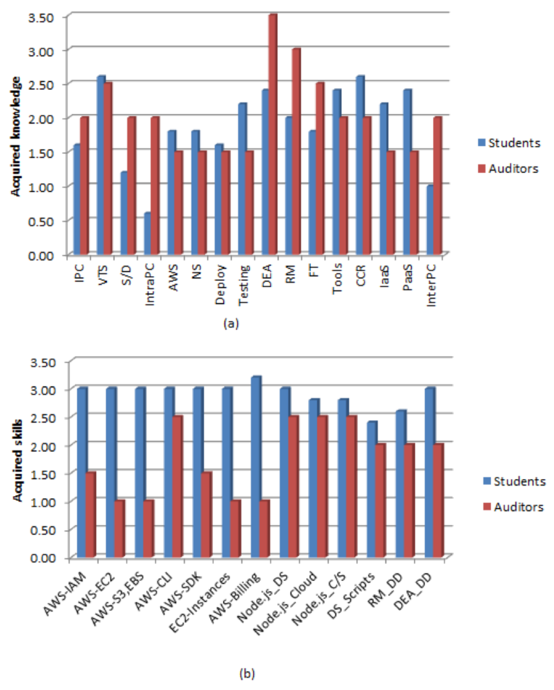


Figure 24. Perception of Acquired Knowledge: Differences Between the After and the Before.

Table 13. Knowledge Survey Questions.

No	**Knowledge/Concept**	**Acronym**
Q1.1	Inter-process communications patterns, like Request-Reply, Request-Reply Acknowledge, and Reliable Multicasts	IPC
Q1.2	Partial ordering of events in a distributed system, as represented by mechanism like Vector Timestamps	VTS
Q1.3	Message serialization/deserialization	S/D
Q1.4	Intra-process concurrency	IntraPC
Q1.5	Computation resources in a cloud-computing environment, such as AWS	AWS
Q1.6	Namespaces, name services, and name resolution	NS
Q1.7	Deployment, execution and testing techniques in a distributed environment	Deploy
Q1.8	Deployment, execution and testing techniques in the cloud.	Testing
Q1.9	Distributed election algorithms	DEA
Q1.10	Resource managers	RM
Q1.11	Fault tolerance in a distributed environment.	FT
Q1.12	Tools for provisioning collection of resources needed for a distributed system.	Tools
Q1.13	Cloud Computing resources	CCR
Q1.14	Infrastructure as a Service (IaaS)	IaaS
Q1.15	Platform as a Service (PaaS)	PaaS
Q1.16	Inter-process concurrency	InterPC

Table 14.Skills Survey Questions.

No	**Skill**	**Acronym**
Q2.1	AWS Users and key pairs (Identity and Access management -IAM)	AWS-IAM
Q2.2	AWS Virtual PC Instances (EC2)	AWS-EC2
Q2.3	AWS Storage (S3, EBS)	AWS-S3,EBS
Q2.4	AWS-CLI (Command Line Interface)	AWS-CLI
Q2.5	AWS SDK (Software Development Kit)	AWS-SDK
Q2.6	Managing instances in AWS: creating/launching, starting, stopping, terminating	EC2-Instances
Q2.7	AWS Billing	AWS-Billing
Q2.8	Using Node.js to Develop Distributed Systems	Node.js_DS
Q2.9	Using Node.js to deploy and run Distributed Systems in the cloud	Node.js_Cloud
Q2.10	Designing and developing TCP/UDP/Web Services-based systems with Node.js	Node.js_C/S
Q2.11	Writing scripts to Deploy/execute applications in distributed environments	DS_Scripts
Q2.12	Designing and Developing Resource Managers	RM_DD
Q2.13	Designing and Developing Distributed Election Algorithms	DEA_DD

Next, notice that the estimated pre-class knowledge levels are higher than the estimated pre-class skill levels. In general, the students felt they had a conceptual understanding of the course concepts, including AWS, which only one student had exposure to before class. From this, we can see that students (and perhaps all people) tend to believe that they are able to generalize conceptual knowledge into new areas that they have not seen before.

Figure 22 shows evidence that the first group of students truly improved their skills. Their net change for every skill was higher than the net change for the second group. Interestingly, the same is not true in the knowledge area. At first glance, this might seem odd, but considering the timing and relative nature of the self-made estimates, there is a possible explanation. Specifically, the students who did not do the assignments naturally felt that their biggest growth was in increase of conceptual knowledge.

4.6. Related Work

Other higher-education institutions are using cloud computing resources in courses that focus on distributed applications or network programming. Clearly, these platforms allow the students to use realistic testing and production environments. Moreover, there are large research universities that have implemented private clouds on their campuses and use them in the classroom. For example, Syracuse University provides a local virtual machine lab used to form virtual networks for security projects [32]. North Carolina State University supplies computing resources over the Internet with their Virtual Computing Lab [33], Arizona State University developed V-lab for Networking Courses [35], and Okanagan College and King's university College talk about using a cloud for educational collaboration [29]. Nevertheless, these private solutions are often not economically viable for many universities [28], and therefore they can only consider public cloud solutions.

Programming assignments that use public or private clouds can add value to the learning experience and increase students' skills directly related to possible professional careers [125] in network programming [28], distributed systems [29], systems administration [125], security [125] [32], data processing [30], among others. Furthermore, a major benefit is that students do not need to simulate network communications over a localhost interface [28]; instead, they can use multiple virtual machines and real network communications to better understanding the distributed system components, their roles, and the related concepts.

Using a public cloud for hands-on activities offers benefits such as scalability, flexibility, security, cost-efficiency and accessibility [28], which all are key characteristics of distributed systems [69]. Public clouds also add an interesting and valuable dimension to the execution and debugging of distributed applications [30], without needing huge budgets for private-cloud or physical-machines infrastructure. Most of the public cloud providers, e.g., Amazon, Google, Microsoft, IBM, offer grants for academic institutions that want to use their resources for educational purposes. For example, at the time of this study, Amazon offered grants up to $100 per students [128]. Other benefits to public clouds include ready access to different operating-system platforms, communication protocols, development tools, open-source code, public forums, and more.

4.7. Conclusions and Future Work

For this small case study, we conclude that programming assignments with requirements to use cloud resources were successful in helping the CS 6200 students to improve their A2R skills, as well as their core distributed-application development skills. Both the instructor's observations and the post-class survey provide anecdotal evidence of their improvement.

We also found some evidence that students are willing and even excited to learn new tools and skills, especially if they can see how it lets them put theory into practice. Even though the assignments were based on carefully crafted and sanitized requirements, they were realistic enough for the students to experience real problems and see how theoretical concepts, like vector timestamps and distributed election, could be used to solve those real problems.

Some important design criteria for assignments included: a) hiding unnecessary details, like all the other capabilities of an EMR beside the generation of disease notifications, b) focusing on requirements that put theory into practice, like the election of an RM in Assignment 4, c) including non-trivial non-functional requirements, like scalability, and d) wherever possible allow students to reuse components or knowledge acquired in previous assignments.

The survey data also opened some doors to possible future research. Specifically, we would like to conduct a broader experiment across multiple software-engineering classes of various kinds and at different levels, to explore specific ways that the design of assignments can improve A2R skills in general. From that, we hope to publish more concrete guidelines for programming-assignment design for software-engineering classes at all levels.

CHAPTER 5

IMPROVING COMPUTER SCIENCE EDUCATION THROUGH CLOUD COMPUTING: AN OBSERVATIONAL STUDY[8]

5.1. Abstract

In an earlier study, we observed that students in a small graduate class who used Cloud Computing (CC) for their programming assignments improved their analysis-to-reuse (A2R) skills more than students who did not use CC. That preliminary result motivated us to see if the use of CC in programming assignments would yield similar results for a broader range of classes and students. To this end, this paper reports on an observational study on the students of the Computer Science Department of Utah State University that spanned from August 2015 to December 2016 and included over 221 students, with data collected at three different times. An ANOVA statistical analysis of the study data revealed a significant difference in the perceptions about acquired A2R skills in favor of students that used CC.

5.2. Introduction

Cloud Computing (CC) is changing the way IT users consume computing resources. CC introduces new elements for execution and development environments, these resources are consumed and shared on demand among software systems stakeholders. Currently, higher education institutions are including CC in their curricula to improve their students' skills [24], and careers have been created around this

[8] Jorge Edison Lascano, Stephen W. Clyde

technology [129] [130] [131] [132] [133] that many organizations have adopted or are moving towards its adoption [134]. CC is currently used in different areas such as infrastructure provisioning, test and development, file storage, disaster recovery, and backups [135]. Most of the industry already requires professionals to have a foundation in the CC body of knowledge, adopted in 2008 by IEEE [136]. This trend represents both an opportunity and a challenge for higher education. Specifically, higher education needs to (a) incorporate CC technology into curricula so students can be better prepared, (b) understand its impact in higher education [137], and (c) leverage CC technology as means of helping students improve other skills and knowledge required for Computer Science (CS) professionals.

Although there exist a wide range of literature in this aspect [27] [31] [32] [35] [138] [33] [29] [139] [140] [141] [142] [143] that cover the use of CC for different courses and the inclusion of CC as a solution for higher-education infrastructure, none of them use CC throughout the full software development process nor do they do not analyze the impact of CC on improving students' A2R skills using CC resources for programing assignments.

CS departments need to improve their students' A2R skills to keep pace with industry demands. Also, they want to comply with accreditation criteria. In the United States of America, the Computing Accreditation Commission of ABET (Accreditation Board for Engineering and Technology, Inc.) [144] is responsible of accrediting the CS programs. ABET establishes general criteria that apply to all programs, and program criteria that apply to a specific program. These general criteria cover program educational objectives (PEO), student outcomes, continuous improvement, curriculum, faculty, facilities, and institutional support. The CS Department of Utah State University has defined the following as one of its PEOs:

PEO-1: "The USU Computer Science program will prepare its graduates to be successful and contributing professionals by being able to apply the principles of computer science and adapt emerging technologies to analyze and solve real world problems" [25].

A preliminary study conducted in 2015 [24], where the instructor of CS6200 from Utah State University used Amazon Web Services for advanced distributed systems programming assignments, showed that CC helps students improve A2R skills. The results of this preliminary study motivated us to plan a new study, encompassing as much of the CS Department as possible, to see if the conclusions would hold for a broader group and different classes. Section 2 provides necessary background information for this observational study and Section 3 summarize a brief literature review on the use of CC in educational institutions.

This observational study aims to find statistically significant differences in the perceptions of students with respect to their A2R skills between those who used CC in their assignments (group A) and who did not (group B). To this end, it focused on two questions: (1) what are students' perceptions about how their own skills and knowledge change over the course of a semester and (2) how does the integration of CC into programming assignments affect those perceptions? Section 4 describes the study's design and the survey instrument used in the study to gather data about student perceptions relative to these two questions. Section 5 provides details about the results of the quantitative analysis. Finally, section 6 presents our conclusions and future research.

5.3. Background

5.3.1. Cloud Computing Services

CC, which “provides shared computer processing resources and data” [145], is a technology trend that is taking off, especially since CC providers now offer services that help organizations overcome their security and compliance concerns [146]. CC is widely becoming an integral piece in the software development process and an important component in complete software solutions. Even IEEE has considered the importance of CC in professional and academic environments and founded in 2011 a global initiative, IEEE Cloud Computing, to promote CC and its related technologies [147]. This initiative comprises standards, publications, education, careers and conferences.

CC providers offer their services under different models [145], three are the standard models as defined by the National Institute of Standards and Technology (NIST): Infrastructure as a Services (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS is used for deployment and execution of computer systems, IaaS can greatly simplify an organization’s IT management and support on-demand scalability, high availability, fault tolerance, and disaster recovery [148]. PaaS provides developers with full out-of-the-box development environments, PaaS solutions typically include virtual machines, operating systems, programming tools, and databases [149]. A third category of CC services is SaaS, solutions in this category are target as end users and cover a wide range of application software. The SaaS users do not need to purchase or install application software locally. Instead, they use existing applications in the cloud. CC services provided by AWS [150] include: virtual instances - EC2 (IaaS), code repositories - CodeCommit (SaaS), code test and deployment –CodePipeline (SaaS), Databases -RDS and DMS (PaaS), scalable storage -S3 (IaaS), big data processing – EMR (PaaS), and web scalability - AWS Elastic Beanstalk (IaaS). Providers such as Google, IBM, Microsoft, and others offer similar CC solutions.

Since cloud services are widely used by enterprises, cloud providers offer their services to educational institutions at no cost or at very low cost, so the new professionals can learn, and test their services. Examples of these programs include Amazon Education [151], Google for Education [152], GitHub Education [153], IBM Cloud Education [154], and Azure in Education [155]. Most of them provide biannual grants to students or instructors.

5.3.2. Observational Studies

Experiments are used to search for cause and effect relationships [156]. Researchers design experiments to predict what can happen by varying some values and observing changes. Variables are used to affect and to quantify these changes and may be any factor, trait or condition. Two important types of variables are independent and dependent variables. The former are changeable or controlled by the scientists. The latter represent the outcome in function of the independent variables [157]. An observational study [158] draws inferences from a sample to a population where the researcher does not have full control over the independent variables [159]. Some reasons for using an observational study instead of a controlled experiment include: the need to respect human rights and logistical issues. Also, sometimes it is simply impossible to control the independent variables sufficiently.

As mentioned before, certain independent variables will be outside the control of the researcher, but they can be observed and recorded. Cause and effect are difficult to establish in observational study, nevertheless, they can allow researchers to formulate some associations and lay the foundation for future studies that can be carried on in control settings [160]. According to Shull et al. [161], after initial feasibility studies, researchers can use observational studies to collect data that will help explain the considered phenomenon and "formulate hypotheses to be tested in subsequent experiments" [162]. "A well designed observational study, resembles, as closely as possible, a simple randomized experiment" [158]. The main difference is the randomization of an experiment, where participants are selected by chance, so bias can be reduced.

5.4. Related Work

The use of virtual resources for CS courses is not a new concept in the academia [27]. Studies report that the use of CC in the class has proofed to be worthwhile, by allowing students to improve their professional skills and to obtain a better understanding of realistic execution environments and issues in areas such as security [31] [27], networking/network programming [32] [35], system and network administration [27] [138], distributed systems [33], and data processing [29] [30]. Other researchers have investigated the relationship between CC and higher education, including CC adoption and its influence [139] [140], students perception of CC effective use [141], CC impact

[142], and relevance of CC [143].

Gonzalez et al. for example, at Rochester Institute of Technology used Amazon EC2 in their Principles of Systems Administration course to leverage career opportunities for their students [27]. Zhu used cloud resources in the Network Programming course at Metropolitan State University of Denver, where students agreed that using CC resources had a positive impact on their learning experience [28], Zhu claims that the same effect will be true for other courses, unfortunately, there is no other studies published to date. Rabkin et al. [30] used CC for MapReduce measurements at University of California, Berkeley, concluding that it is a need for students to experience running and debugging distributed applications in a realistic infrastructure. None of them have used CC as a resource for programming assignments.

5.5. Design of the Observational Study

In [24], we described our findings on the use of Cloud resources for programming assignments. Students that used CC for their assignments improved their A2R skills in a level higher than those ones that did not use CC. This group of students corresponds to the CS6200 course taught during Spring-2015, where participants developed advanced distributed applications, some of them used IaaS and PaaS, others did not use it. After this experience, we aimed to extend our work to other programming courses, by including CC as a resource for students' assignments during Spring-2016 and Fall-2016 semesters. With the hoped-for a bigger improvement on A2R skills for these students than for the Fall-2015 students, who did not use CC resources.

We defined three main variables for our experiment, the use of CC (the treatment) as the independent variable, and the variations in A2R skills and knowledge levels (covariates) as the dependent variables. Nonetheless, it was not possible to prepare a properly randomized selection of students who will or will not use CC in assignments at the CS department. So, the independent variable is outside our direct control. Alternatively, we requested the instructors of CS3450, CS5110, CS5200, CS5600, CS5680, CS5700, CS5800, CS5890, CS6110, CS6600, and CS7910 courses to use CC in their assignments. During the Spring-2016 semester, the instructor of CS5200 agreed to use CC, and during the Fall-2016 semester, the instructors for CS3450, CS5600, CS5700, and CS6600 agreed to the request. Students in all other classes would continue not to use CC in their programming assignments.

The most of the CC services that ended being used were IaaS, such as AWS EC2 virtual servers and Amazon S3 storage. Services classes did used some simple SaaS services, like Bitbucket and GitHub, for Git repositories. Requiring instructors to use CC was not possible and following up with the students who used CC was challenging. So, we let the instructors use CC on their discretion and made sure the survey instrument would collect sufficient information to determine whether they used CC in their programming assignments, as well as their perceptions about their levels of skill and knowledge.

Furthermore, we designed the surveyed instrument overall curriculum objectives in mind, like PEO-1 and the following student outcomes from ABET's guidelines [163]:

(b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution

(c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs

(i) An ability to use current techniques, skills, and tools necessary for computing practice.

(k) An ability to apply design and development principles in the construction of software systems of varying complexity.

To organize the survey questions, we first decomposed PEO-1 into three sub-objectives: Analyze and solve real world problems, adapt emerging technologies, and apply principles of CS (see Figure 25). The first two sub-objectives deal with A2R skills level, while the third deals with general knowledge. Next, we aligned the sub-objectives with the student outcomes and then decomposed the outcomes into eight areas: Problem Analysis, Requirements Identification and Definition, Systems Analysis, Systems Design, Current Practices, Other Software Engineering Skills, Tools, and Principles. The questions were designed with these eight areas in mind, following the Goal Question Metric paradigm [164].

The first four questions of the survey though, are not meant to measure any level of knowledge or skills, they are intended to capture the characteristics of the participant and their classes, i.e., student number, age, past classes, and current classes.

Question 5 captures the students' perception of A2R skills level, while Q6 measures the students' perception of knowledge. Each of the questions contain a number of sub-questions or topics and asked the participants to rank their perceptions of their own skills and knowledge on a scale from 1 to 5, relative to both the beginning and at the end of every period, for each one. Each topic has a *N/A* option for students who do not recognize the topic or feel that it does not. See Figure 26 for a snippet of Question 5 that shows just two topics.

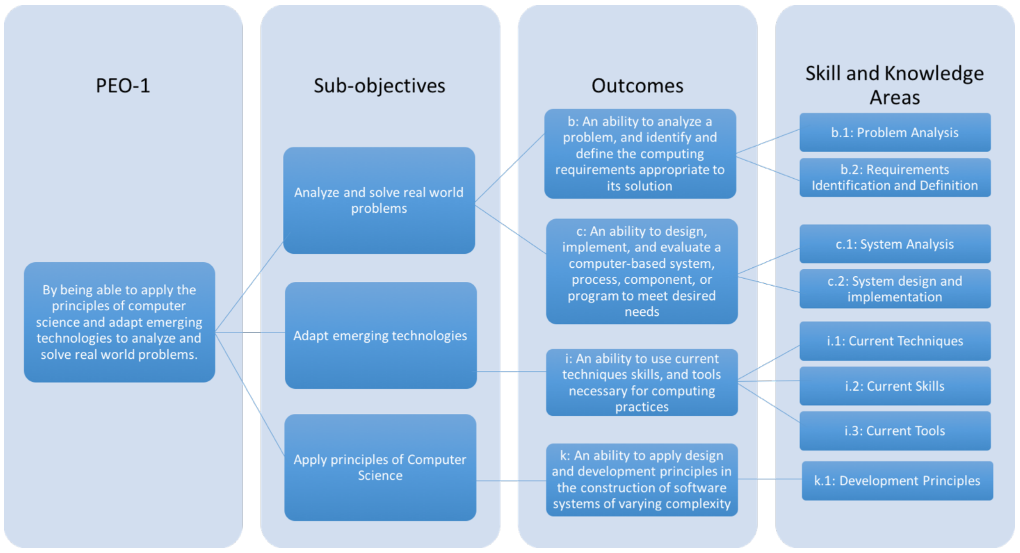
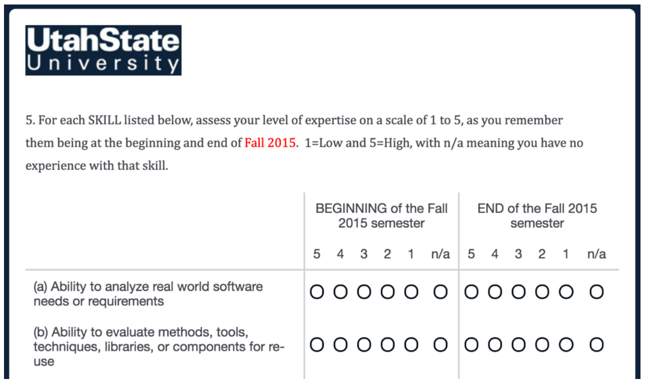

PEO-1
Sub-objectives
Outcomes
Skill and Knowledge Areas
By being able to apply the principles of computer science and adapt emerging technologies to analyze and solve real world problems.
Analyze and solve real world problems
Adapt emerging technologies
Apply principles of Computer Science
b: An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution
c: An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs
i: An ability to use current techniques skills, and tools necessary for computing practices
k: An ability to apply design and development principles in the construction of software systems of varying complexity
b.1: Problem Analysis
b.2: Requirements Identification and Definition
c.1: System Analysis
c.2: System design and implementation
i.1: Current Techniques
i.2: Current Skills
i.3: Current Tools
k.1: Development Principles

Figure 25. Goal Question Metric Applied to PEO-1


UtahState University

5. For each SKILL listed below, assess your level of expertise on a scale of 1 to 5, as you remember them being at the beginning and end of Fall 2015. 1=Low and 5=High, with n/a meaning you have no experience with that skill.

	BEGINNING of the Fall 2015 semester						END of the Fall 2015 semester					
	5	4	3	2	1	n/a	5	4	3	2	1	n/a
(a) Ability to analyze real world software needs or requirements	○	○	○	○	○	○	○	○	○	○	○	○
(b) Ability to evaluate methods, tools, techniques, libraries, or components for re-use	○	○	○	○	○	○	○	○	○	○	○	○

Figure 26. Snippet of Question 5 from the Fall-2015 Semester Survey

Question 7 is about design principles and assignments characteristics. Questions 8 and 9 of the survey were open so the students can express themselves about how they think that their skills can be improved and about what methods or tools would help them to adapt to emerging technologies.

The study design originally called for a survey to be conducted using this instrument at the end of every semester throughout the study period. However, because of semester breaks, it became necessary to administer the surveys at the beginning of the following semester.

While planning the study we applied for and obtained the necessary Institutional Review Board (IRB) approvals, and then started collecting data from the students of the CS department of Utah State University at the end of three contiguous semesters: Fall-2015, Spring-2016, and Fall-2016 semesters. The study intended to reach about 650 students of the CS department, i.e. students that took CS classes in any of the Fall-2015, Spring-2016 and Fall-2016 semesters. From those students, a total of 221 responded to the different surveys, this allowed us to have some answers in the different semesters for the same students. For the Fall-2015 semester, there were 153 respondents to the survey, for Spring-2016, 181 and for Fall-2016, 117. Before we started processing the data, we needed to clean it. Incomplete surveys were dropped. Meaningless survey data such as answers with the same level of perception for all questions were eliminated. Surveys with duplicated students' id numbers for the same semester were also removed.

Significant tests computed the probability for our hypothesis, H_1: Students who had assignments that required the use of CC have a greater perceived increase in her/his skills and knowledge levels in the following A2R areas: tool evaluation, development environment setup, runtime environment configuration, analysis, design, application of "best practices", testing, deployment, and reuse.

These questions/metrics (Table 16, Table 15) capture students' A2R skills level in a scale from 1 to 5 at the beginning and end of a semester, the difference ΔA2R = $A2R_{After} – A2R_{Before}$ represents students' capacity variation on their skills in the course of a semester. To have a total measurement of A2R skills composed by the 23 sections of question 5, we gave arbitrary weights to every section based on their importance to the outcome they are dealing with, i.e. to the ABET PEO-1 decomposition, and to industry requirements.

Table 15. Weights for Question 6: Perceptions of Level of Students' Knowledge

#	*Knowledge Question*	*Weight*
a	Understanding of how to match the needs of an application to an appropriate develop environment and runtime platform	0.8
b	Understanding of at least one development stack (a collection of reusable components or libraries)	0.7
c	Understanding of development environment setup	0.6
d	Understanding of principles of software testing	0.8
e	Understanding of network communications	0.4
f	Understanding of the principle of reliability as it applies to software systems	0.7
g	Understanding of security principles and practices for software systems	0.7
h	Understanding of what affect runtime performance and how to detect inefficiencies and correct them	0.5
i	Understanding of the principle of Coupling	0.5
j	Understanding of the principle of Cohesion	0.5
k	Understanding of software reuse	0.8
l	Understanding "abstraction" with respect to the design and implementation of software system.	0.6
m	Understanding "encapsulation" with respect to the design and implementation of software systems	0.6
n	Understanding "modularization" with respect to the design and implementation of a software system	0.6

The main purpose of this observational study is to analyze the impact of using CC in programming assignments for improving PEO-1. For this reason, we compared statistically ΔA2R's of students that use CC (group A), and students that do not use CC (group B), $\Delta A2R_A$ vs $\Delta A2R_B$. Finally, we define our null hypothesis, H_0, that states: *The use of Cloud Computing resources for programming assignments does not have any effect on the increment of A2R skills on students*.

Table 16. Weights for Question 5: Perceptions of Level of Students' A2R Skills

#	***A2R Question***	***Weight***
a	Ability to analyze real world software needs or requirements	0.6
b	Ability to evaluate methods, tools, techniques, libraries, or components for re-use	0.1
c	Ability to reuse methods, designs, or software from previous assignments	0.8
d	Ability to design a software system to meet complex real-world requirements	0.8
e	Ability to implement a software system according to a design	0.8
f	Ability to implement software that can run in a runtime environment different from your own computer or one in a school lab	0.4
g	How to configure a runtime environment into which you can deploy a software system that you built	0.6
h	Ability to deploy a system to a runtime environment different from your own computer or one in a school lab	0.3
i	Ability to learn and use virtual or cloud-based resources for creating software solutions	0.6
j	Ability to thoroughly test software systems using executable test cases	0.8
k	Ability to think critically and develop alternative solutions to a problem	0.6
l	Ability to think creatively about software solutions	0.8
m	Ability to understand, evaluate and use emerging technologies	0.4
n	Ability to learn and use existing software services available on the Internet	0.4
o	Ability to follow industry-wide "best practices" when use your chosen development environment	0.5
p	Ability to follow "best practices" in testing	0.8
q	Ability to follow "best practices" in deploying software to runtime environments	0.4
r	Ability to apply green practices	0.1
s	Ability to evaluate a variety of operating systems and frameworks as possible runtime environments	0.1
t	Ability to apply collaborative methods, tools, techniques to develop software	0.6
u	Ability to design a software system that will provide for a good user experience	0.6
v	The ability to create maintainable software	0.6
w	Ability to understand real-world problems related to the course material	0.8

5.6. Data Analysis and Interpretation

The data collected throughout the three semesters span was compared from different perspectives and for different groups of students. For every comparison, an analysis of variance (ANOVA) was performed using ANOVA for a single factor, t-test or z-test. We gathered a total of 451 responses from the three surveys, 280 were answered in their entirety, and 264 were considered valid, 95 for Fall-2015, 102 for Spring-2016, and 67 for Fall-2016. From the 264 students, only 37 would have taken courses that include the use of CC in their programming assignments, leaving a total of 227 students that did not have exposure to CC. There are 16 surveyed students that took either CS5200 or CS5700 when CC was not used in their assignments during Fall-2015; and there are 12 students that took CS5200 or CS5700 when CC was introduced as a resource for programming assignments during Spring-2016 and Fall-2016. Table 17 summarizes the values statistically obtained, namely: Averages, p-values, and Differences.

We run a qualitative analysis by gathering data about the level of perception of every student and assigning an arbitrary weight to every question. We relied on the analysis of variance to proof the nonvalidity of our Null Hypothesis, H_0.

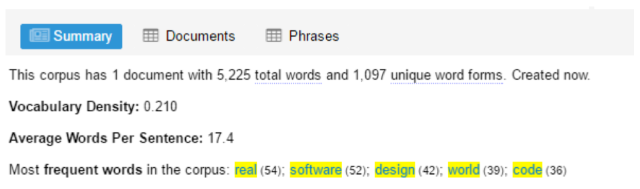
Table 17. ANOVA P-values for A2R Skills Variations

Comparison	**Groups**	**Count**	**Average**	**P-value**	**Diff.**	**YES/NO statistically significant difference**
ΔCC_A vs ΔCC_B	Q5i GA diff Q5i GB diff	37	1.1622	0.0215	0.4573	YES
	Q5 S16 Before	227	0.7048			
$(A2R_A$ vs $A2R_B)_{End}$	Q5 GA End	37	3.3366	0.2477	0.1692	NO
	Q5 GB End	227	3.1674			
$(A2R_A$ vs $A2R_B)_{Beginning}$	Q5 GA Beg	37	2.3523	0.5424	-0.0967	NO
	Q5 GB Beg	227	2.4491			
$\Delta A2R_A$ vs $\Delta A2R_B$	Q5 GA diff	37	0.9842	0.0141	0.2621	YES
	Q5 GB diff	227	0.7220			
$\Delta A2R_{A\text{-}5200/5700}$ vs $\Delta A2R_{B\text{-}5200/5700}$	Diff A2R NoCC	16	0.6775	0.0096	0.4265	YES
	Diff A2R CC	12	1.1041			
Same Students F15 – S16	Q5 F15 After	30	3.2056	0.0005	-0.6927	YES
	Q5 S16 Before	30	2.3843			
Same Students S-16 – F16	Q5 S16 After	16	3.1882	0.0633	-0.8213	NO
	Q5 GB diff	227	0.7220			

For this case study, we concluded that students that use CC resources for their programming assignments improved the A2R skills in a higher level than students that did not use it, obtaining an improvement on PEO-1. This inference is supported by calculating the probability for H_0. We analyzed the differences between group A and group B ($\Delta A2R_A$ *vs* $\Delta A2R_B$). We got a p-value = 0.014; as p-value < 0.05, H_0 is rejected in favor of H_1.

Students of CS5200 and CS5700 courses were special groups that in previous semesters (S_{i-1}) did not use CC in their assignments and in following semesters (S_i) use CC resources. Their perceptions show similar results than the comparison between groups A and B. with a well noted tendency to a higher increase, 0.426, for group $A_{5200/5700}$, versus 0.262 for group $B_{5200/5700}$. In this analysis H_0 was strongly rejected in favor of H_1 with a p-value=0.00959 < 0.05, that is a positive influence of using CC improve students' A2R skills. These results corroborate the rejection of H_0, nevertheless a different environment setup may be needed for future experiments where a group of students use CC and other group in the same class do not.

Besides the quantitative analysis, there is the possibility for a qualitative analysis based on Questions 8 and 9. We found out after a brief text analysis of Question 8 that *Real, Software, Design, World* are the most used words, see Figure 27. After this word count analysis run using the Voyant tools [165], a sentiment analysis may be the next step to infer some conclusions in respect to positive or negative sentiments relative to PEO-1. A sentiment analysis is not part of this research.



This corpus has 1 document with 5,225 total words and 1,097 unique word forms. Created now.

Vocabulary Density: 0.210

Average Words Per Sentence: 17.4

Most **frequent words** in the corpus: real (54); software (52); design (42); world (39); code (36)

Figure 27. Word Count Analysis for Question 8 Using Voyant-tools

5.7. Conclusions and Future Directions

In this paper, we presented the results of a one-and-half year observational study about the use of CC in programming assignments for a group of CS students, concluding that the use of CC has a positive impact. Specifically, our findings confirmed our hypothesis that students exposed to CC in their assignments have a bigger increase in their perception of acquired A2R skills than students who did not use any CC in their assignments. Therefore, there is a likelihood that the use of CC in programming assignments for CS students has a positive influence in increasing students' A2R skills.

We tested for significance using the Analysis of Variance Algorithm (ANOVA [166]) with a probability (p-value) cutoff of 0.05. In most cases, we obtained a p-value < 0.05 meaning that there is statistically significant difference [167] [168] between the group that used CC and the group that did not use CC. Hence, we rejected H_0 in favor of the alternate hypothesis, H_1, i.e., students who were exposed to CC ($\Delta A2R_A$) noticed a bigger increase in their perceptions of acquired A2R skills than students that were not exposed to CC ($\Delta A2R_B$).

This study's findings open doors for future experiments. An interesting hypotheses for follow-on experiments would be "the use of CC can help improve communication skills for team projects assignments." An experiment would consider students in a software-engineering course complete assignments in groups instead of individually. Some course sections would use CC and others would not. A more complex experiment would involve the continuation of projects in a course sequence, e.g., one course could model a system and subsequent courses would then implement that model. A couple of sections would use CC services, such as virtual servers and code repositories, other sections would use nonCC development tools and share code by copying files from computer to computer.

Also future experiments could look at even broader populations and group randomization techniques. For example, group selection would be better if we could split students within a single class into A and B groups. Unfortunately, that may interfere in the learning process or create a lot of extra work for the instructors. For example, we may need to plan ahead recovery courses for students that show a lower increase on their skills through the experiment.

The survey used in this study contained a couple open questions, namely Questions 8 and 9. A future research project could use sentiment analysis [169] to analyze the qualitative data that were collected for these open questions. This could provide some insights for how the students feel about their skills and their knowledge. An informal analysis of these data shows that the most used words are “software”, “real” and “world”. From here, the sentiment analysis would try to determine whether these ideas are connected to positive, negative and neutral attitudes.

Uncertainty of the advantages of CC in education still exists, even though enterprises of all kinds are currently using CC to build and deploying software. One disadvantage, is that for novice developers, like students, CC services are just one more complexity that they must learn and it could easily become a stumbling block. Any integration of CC in a CS curriculum must be well thought out and aim to eliminate or minimize these obstacles.

Since this research was an observational study and we could not control the group selection, we ended up with widely different group sizes: 37 in group A and 227 in group B. Ideally, the size of each group should be the same. Nevertheless, ANOVA allows for uneven group sizes. Other comparison sets were found among the students of contiguous semesters for the same subject. For example, CS5200 and CS5700's students that did not use CC during the semester S_i, i.e., 16 from group B, versus the same courses' students that used CC during the semester S_{i+1}, i.e., 12 from group A. However, this comparison introduction would add extra variables, such as variations in commitment in faculty members and motivation for students to be part of this study.

The adoption of CC for programming assignments in the research population of USU was not widely accepted, instead instructors were inclined to reject its use, a future study could analyze the causes and propose a better approach such as the Technology Acceptance Model (TAM) which focuses in the perceived usefulness and perceived ease-of-use of new technologies.

We performed a statistical analysis of Students' perceptions of A2R skills and knowledge levels at the end of Fall-2015 versus the beginning of Spring-2016. Same analysis was performed at the end of Fall-2016 versus the beginning of Fall-2016. Although we thought that their perception should not be statistically significantly different at the end of semester S_i than at the beginning of semester S_{i+1}, the results, Table 17, showed that they are statistically different. Future studies may help determine better methods to help students retain their knowledge and skills between semesters' breaks.

5.8. Acknowledgement

We thank Seyed Mansourbeigi for assistance with the statistical analysis.

CHAPTER 6

CONCLUSIONS

Gathering the knowledge to define design patterns only con be achieved by professionals that have worked in the field for a long time. This work must be extended by including other professionals, they can endorse the current set of patterns and they could contribute with their own solutions. For this reason we have created and published a wiki especially for contributions on the development and improvement of CommDP.

The next step in this respect is to add more common solutions to CommDP, hence increasing its range and usability. Additionally, as the language is defined and its qualities are studied based on current protocols such as HTTP or FTP, we need to find applicability for patterns. It can be reached from the academia, where students can learn the proven solutions and implement their own protocols. Also it must be proven by the developer community. As future work, we should implement existing communication software using CommDP and measure these implementations' qualities.

The use of CC in programing assignments has proofed to be valuable in increasing A2R skills of CS students that used CC than students that did not use it. Statistical results based on ANOVA allowed to conclude that there is a statistical difference on the perceptions for students that used CC in their programming assignments. Nevertheless, the results may be biased by a few external factors such as the different methods, the different content and the improvement in the methodology that the instructors could have used in following semesters.

To improve the previous study, and following the next phase after an observation, future work could include a wider sample, i.e. students from different CS related careers in different higher education institutions. Also, there is the need of controlling the independent variables such as the random selection of the population. Finally, instead of studying contiguous semester students, as is the case in chapter 5, the universe should be selected from the same group of students to avoid biased influences such as the difference in methodology or content by instructors in different semesters. A better environment would be a set of students of the same course, half using CC and half not using it. Unfortunately, there are some factors that can impede this study to be carried on properly. Students and instructors may not be willing to participate in the experiment, and some of them may be more affected than others. Careful attention should be taken, especially in negative effects over the group that would not use CC.

The two topics covered in this dissertation, CommDP and the use of CC for improving computer science education are intended to be part of a series of publications. The introduction of CommDP [84], presented in chapter 2, has been published in ICSEA 2016 [11], see Appendix A. This paper was awarded as one of the best papers of the conference, consequently we got an invitation to submit an extended article on the IARIA journals [38]. This extended work is being submitted before August 2017 for its publication. On the other hand, the use of CC to improve computer science education has evolved in two papers, the first one [24], covered in chapter 4, has been published in ICSEA 2015 [37], see Appendix B. Its follow up, the results of one-and-a-half year observational study has been submitted to a more appropriate conference, CSEE&T 2017, the 30th Conference on Software Engineering Education and Training.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1 edition. Addison-Wesley Professional, 1994.

[2] M. A. Sheta, M. Zaki, K. A. E. S. E. Hadad, and H. A. M, “Anti-spyware Security Design Patterns,” in *2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC)*, 2016, pp. 465–470.

[3] O. Ur-Rehman and N. Zivic, “Secure Design Patterns for Security in Smart Metering Systems,” in *2015 IEEE European Modelling Symposium (EMS)*, 2015, pp. 278–283.

[4] M. Fowler, *Patterns of Enterprise Application Architecture*, 1 edition. Boston: Addison-Wesley Professional, 2002.

[5] A. G. Leal and M. Boldt, “A big data analytics design patterns to select customers for electricity theft inspection,” in *2016 IEEE PES Transmission Distribution Conference and Exposition-Latin America (PES T D-LA)*, 2016, pp. 1–6.

[6] N. Ivaki, N. Laranjeiro, and F. Araujo, “Towards designing reliable messaging patterns,” in *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, 2016, pp. 204–207.

[7] V. Malcher, “Design Patterns in Management Systems,” in *2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, 2016, pp. 36–39.

[8] T. Wetchakorn and N. Prompoon, “Method for mobile user interface design patterns creation for iOS platform,” in *2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2015, pp. 150–155.

[9] “RightScale 2016 State of the Cloud Report.” [Online]. Available: https://www.rightscale.com/lp/state-of-the-cloud. [Accessed: 07-Dec-2016].

[10] J. Lascano and S. Clyde, “A Pattern Language for Application-level Communication Protocols,” presented at the ICSEA 2016, The Eleventh International Conference on Software Engineering Advances, 2016, pp. 22–30.

[11] “ICSEA 2016.” [Online]. Available: https://www.iaria.org/conferences2016/ICSEA16.html. [Accessed: 09-Dec-2016].

[12] “Distributed computing,” *Wikipedia*. 11-Jan-2017.

[13] L. Rising, *Design Patterns in Communications Software*, 1 edition. Cambridge ; New York: Cambridge University Press, 2001.

[14] L. Li and W. Chou, "Design Patterns for RESTful Communication Web Services," in *2010 IEEE International Conference on Web Services (ICWS)*, 2010, pp. 512–519.

[15] R. Daigneau, *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*, 1 edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2011.

[16] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Science & Business Media, 2014.

[17] F. Buschmann, K. Henney, and D. C. Schmidt, *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*, Volume 4 edition. Wiley, 2007.

[18] "An Architectural Overview of the ACE Framework A Case study of Successful Cross-platform Systems Software Reuse." [Online]. Available: https://www.researchgate.net/publication/250496073_An_Architectural_Overview_of_the_ACE_Framework_A_Casestudy_of_Successful_Cross-platform_Systems_Software_Reuse. [Accessed: 10-Mar-2016].

[19] D. C. Schmidt, "ACE+TAO Overview," *Douglas C. Schmidt*. [Online]. Available: http://www.dre.vanderbilt.edu/~schmidt/TAO-overview.html. [Accessed: 10-Mar-2016].

[20] "Patterns for Distributed Real-time and Embedded Systems." [Online]. Available: https://www.dre.vanderbilt.edu/~schmidt/patterns-ace.html. [Accessed: 04-Mar-2016].

[21] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, Volume 1 edition. Chichester ; New York: Wiley, 1996.

[22] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*, Volume 2 edition. Chichester England ; New York: Wiley, 2000.

[23] M. Kircher and P. Jain, *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*, Volume 3 edition. Chichester England; New York: Wiley, 2004.

[24] J. E. Lascano and S. W. Clyde, “Using Cloud Services To Improve Software Engineering Education for Distributed Application Development,” presented at the ICSEA 2015, The Tenth International Conference on Software Engineering Advances, 2015, pp. 438–444.

[25] “Program Educational Objectives.” [Online]. Available: https://cs.usu.edu/assessment/objectives. [Accessed: 21-Dec-2016].

[26] “The 30th IEEE Conference on Software Engineering, Education, and Training.” [Online]. Available: http://www.cseet2017.com/index.html. [Accessed: 24-May-2017].

[27] C. Gonzalez, C. Border, and T. Oh, “Teaching in Amazon EC2,” in *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education*, New York, NY, USA, 2013, pp. 149–150.

[28] W. Zhu, “Hands-On Network Programming Projects in the Cloud,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2015, pp. 326–331.

[29] Y. Khmelevsky and V. Voytenko, “Cloud Computing Infrastructure Prototype for University Education and Research,” in *Proceedings of the 15th Western Canadian Conference on Computing Education*, New York, NY, USA, 2010, p. 8:1–8:5.

[30] A. Rabkin, C. Reiss, R. Katz, and D. Patterson, “Using clouds for MapReduce measurement assignments,” *ACM Trans. Comput. Educ.*, vol. 13, no. 1, pp. 1–18, Jan. 2013.

[31] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, “Seattle: A Platform for Educational Cloud Computing,” in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, New York, NY, USA, 2009, pp. 111–115.

[32] W. Du and R. Wang, “SEED: A Suite of Instructional Laboratories for Computer Security Education,” *J Educ Resour Comput*, vol. 8, no. 1, p. 3:1–3:24, Mar. 2008.

[33] H. E. Schaffer, S. F. Averitt, M. I. Hoit, A. Peeler, E. D. Sills, and M. A. Vouk, “NCSU’s Virtual Computing Lab: A Cloud Computing Solution,” *Computer*, vol. 42, no. 7, pp. 94–97, Jul. 2009.

[34] W. Sun, V. Katta, K. Krishna, and R. Sekar, “V-NetLab: An Approach for Realizing Logically Isolated Networks for Security Experiments,” in *Proceedings of the Conference on Cyber Security Experimentation and Test*, Berkeley, CA, USA, 2008, p. 5:1–5:6.

[35] L. Xu, D. Huang, and W.-T. Tsai, "V-lab: A Cloud-based Virtual Laboratory Platform for Hands-on Networking Courses," in *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, New York, NY, USA, 2012, pp. 256–261.

[36] ACM Computing Curricula Task Force, Ed., *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc, 2013.

[37] "ICSEA 2015." [Online]. Available: https://www.iaria.org/conferences2015/ICSEA15.html. [Accessed: 09-Dec-2016].

[38] "IARIA Journals." [Online]. Available: http://www.iariajournals.org/. [Accessed: 09-Dec-2016].

[39] "The 30th IEEE Conference on Software Engineering, Education, and Training." [Online]. Available: http://www.cseet2017.com/documents.html. [Accessed: 09-Dec-2016].

[40] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5 edition. Boston: Pearson, 2011.

[41] "Distributed computing," *Wikipedia, the free encyclopedia*. 27-Feb-2016.

[42] S. W. Clyde, "Object mitosis: a systematic approach to splitting objects across subsystems," in *, Proceedings of the Third International Workshop on Object Orientation in Operating Systems, 1993*, 1993, pp. 182–185.

[43] "Communications protocol," *Wikipedia, the free encyclopedia*. 10-Apr-2016.

[44] "protocol | computer science," *Encyclopedia Britannica*. [Online]. Available: http://www.britannica.com/technology/protocol-computer-science. [Accessed: 20-Apr-2016].

[45] J. O. Coplien and N. B. Harrison, *Organizational Patterns of Agile Software Development*. Upper Saddle River, NJ: Prentice Hall, 2004.

[46] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press, 1977.

[47] C. White, *Data Communications and Computer Networks: A Business User's Approach*, 7 edition. Boston, MA: Cengage Learning, 2012.

[48] "ISO/IEC 10026-1:1992 - Information technology -- Open Systems Interconnection -- Distributed Transaction Processing -- Part 1: OSI TP Model." [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=17979. [Accessed: 20-Apr-2016].

[49] K. Beck and W. Cunningham, "Using Pattern Languages for Object-Oriented Programs," in *OOPSLA-87*, Orlando, Florida, 1987.

[50] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*, Volume 2 edition. Chichester England ; New York: Wiley, 2000.

[51] J. Tidwell, *Designing Interfaces*, 2 edition. Sebastopol, CA: O'Reilly Media, 2011.

[52] D. C. Schmidt, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Dispatching," in *Pattern languages of program design*, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 529–545.

[53] I. Pyarali, T. Harrison, and D. Schmidt, "Asynchronous Completion Token: an Object Behavioral Pattern for Efficient Asynchronous Event Handling," in *3rd Annual Conference on The Pattern Languages of Programs*, Monticello, Illinois, 1997, pp. 1–7.

[54] I. Pyarali, T. Harrison, D. C. Schmidt, and T. D. Jordan, *Proactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handlers for Asynchronous Events*. 1997.

[55] M. K. Douglas C. Schmidt, "Leader/Followers - A Design Pattern for Efficient Multi-threaded Event Demultiplexing and Dispatching," in *7th Pattern Languages of Programs Conference*, Allerton Park, Illinois, 2000.

[56] D. C. Schmidt and T. Harrison, "Double-checked locking," in *Pattern languages of program design*, vol. 3, 1997, pp. 363–375.

[57] L. Rising, *Design Patterns in Communications Software*, 1 edition. Cambridge ; New York: Cambridge University Press, 2001.

[58] P. Jain and D. C. Schmidt, "Service Configurator: A Pattern for Dynamic Configuration of Services," in *Proceedings of the 3rd Conference on USENIX Conference on Object-Oriented Technologies (COOTS) - Volume 3*, Berkeley, CA, USA, 1997, pp. 16–16.

[59] R. C. Martin, D. Riehle, and F. Buschmann, *Pattern Languages of Program Design 3*, 1 edition. Reading, Mass: Addison-Wesley Professional, 1997.

[60] D. C. Schmidt, "Using design patterns to develop reusable object-oriented communication software," *Commun. ACM*, vol. 38, no. 10, pp. 65–74, 1995.

[61] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1 edition. Addison-Wesley Professional, 1994.

[62] 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.

[63] "Modularity," *Wikipedia, the free encyclopedia*. 11-Apr-2016.

[64] M. Burrows, "The Chubby Lock Service for Loosely-coupled Distributed Systems," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2006, pp. 335–350.

[65] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[66] M. Raynal, "About Logical Clocks for Distributed Systems," *SIGOPS Oper Syst Rev*, vol. 26, no. 1, pp. 41–48, Jan. 1992.

[67] F. Mattern, "Virtual time and global states of distributed systems," in *Parallel and Distributed Algorithms*, 1989, pp. 215–226.

[68] C. Fidge, "Logical Time in Distributed Computing Systems," *Computer*, vol. 24, no. 8, pp. 28–33, Aug. 1991.

[69] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th. edition. Boston: Pearson, 2011.

[70] H. T. Kung and J. T. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans Database Syst*, vol. 6, no. 2, pp. 213–226, Jun. 1981.

[71] M. Voelter, "Patterns for Handling Cross-cutting Concerns in Model-Driven Software Development," in *ResearchGate*, 2005.

[72] A. S. Tanenbaum and D. Wetherall, *Computer networks*, 5th ed. Boston: Pearson Prentice Hall, 2011.

[73] R. Daigneau, *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*, 1 edition. Upper Saddle River, NJ: Addison-Wesley Professional, 2011.

[74] S. N. Bhatti, “Why Quality?: ISO 9126 Software Quality Metrics (Functionality) Support by UML Suite,” *SIGSOFT Softw Eng Notes*, vol. 30, no. 2, pp. 1–5, Mar. 2005.

[75] R. Rambo, P. B. Ph.D, and E. Branyan, “Establishment and Validation of Software Metric Factors,” *J. Parametr.*, vol. 5, no. 3, pp. 21–32, Sep. 1985.

[76] G. Bochmann and C. Sunshine, “Formal Methods in Communication Protocol Design,” *IEEE Trans. Commun.*, vol. 28, no. 4, pp. 624–631, Apr. 1980.

[77] R. Sharp, *Principles of Protocol Design*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[78] C. Dendorfer and R. Weber, *Development and Implementation of a Communication Protocol: An Exercise in FOCUS*. Mathematisches Institut and Institut für Informatik der technischen Universität München, 1992.

[79] J. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, “CADP a protocol validation and verification toolbox,” in *Computer Aided Verification*, 1996, pp. 437–440.

[80] M. Fowler, *Patterns of Enterprise Application Architecture*, 1 edition. Boston: Addison-Wesley Professional, 2002.

[81] K. Lochmann and A. Goeb, “A unifying model for software quality,” in *Proceedings of the 8th international workshop on Software quality*, 2011, pp. 3–10.

[82] “ISO/IEC 25010:2011 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.” [Online]. Available: https://www.iso.org/standard/35733.html. [Accessed: 01-May-2017].

[83] “ISO 25010.” [Online]. Available: http://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0. [Accessed: 28-Apr-2017].

[84] J. E. Lascano and S. Clyde, “A Pattern Language for Application-level Communication Protocols,” presented at the ICSEA 2016, The Eleventh International Conference on Software Engineering Advances, 2016, pp. 22–30.

[85] Y. Y. Nasrallah, I. Al-Anbagi, and H. T. Mouftah, “A quality of service model for IEEE 802.11p communication protocol in a smart city,” in *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, 2014, pp. 1–3.

[86] W. Zhang, K. M. Hansen, J. Fernandes, J. Schuette, and F. M. Lardies, “QoS-Aware Self-adaptation of Communication Protocols in a Pervasive Service Middleware,” in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int’l Conference on Int’l Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010, pp. 17–26.

[87] A. Coccoli, A. Bondavalli, and F. D. Giandomenico, “Analysis and estimation of the quality of service of group communication protocols,” in *Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISORC 2001*, 2001, pp. 209–216.

[88] J. E. Lascano and S. Clyde, “Communication-protocol Design Patterns (CommDP) - COMMDP.” [Online]. Available: http://commdp.serv.usu.edu/wiki/index.php/Communication-protocol_Design_Patterns_(CommDP). [Accessed: 13-Jan-2017].

[89] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, 1 edition. Boston: Addison-Wesley Professional, 2003.

[90] “Transport Message Exchange Pattern.” [Online]. Available: https://www.w3.org/2000/xp/Group/1/10/11/2001-10-11-SRR-Transport_MEP. [Accessed: 25-May-2017].

[91] “Echo (communications protocol),” *Wikipedia*. 19-Jun-2016.

[92] J. Postel, “Echo Protocol.” [Online]. Available: https://tools.ietf.org/html/rfc862. [Accessed: 21-Feb-2017].

[93] “Simple Network Management Protocol,” *Wikipedia*. 18-Jan-2017.

[94] R. T. Fielding, T. Berners-Lee, and H. Frystyk, “Hypertext Transfer Protocol -- HTTP/1.0.” [Online]. Available: https://tools.ietf.org/search/rfc1945. [Accessed: 21-Feb-2017].

[95] K. A. Robbins and S. Robbins, *UNIX Systems Programming: Communication, Concurrency, and Threads*. Prentice Hall Professional, 2003.

[96] “Semantics.” [Online]. Available: http://www.cs.unc.edu/~dewan/242/s04/notes/ipc/node30.html. [Accessed: 07-Mar-2017].

[97] “11 UDP Transport — An Introduction to Computer Networks, edition 1.9.0.” [Online]. Available: http://intronetworks.cs.luc.edu/current/html/udp.html. [Accessed: 21-Feb-2017].

[98] N. Halder, A. T. Islam, and J. B. Song, “Modeling and Formal Verification of Communication Protocols for Remote Procedure Call,” *IJCSNS*, vol. 7, no. 7, p. 63, 2007.

[99] K. H. Talukder and K. Harada, “Modeling and verification of some communication protocols,” in *2006 8th International Conference Advanced Communication Technology*, 2006, vol. 3, p. 6 pp.-2198.

[100] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP).” [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 17-Feb-2017].

[101] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, and S. Reddy, *HTTP: The Definitive Guide: The Definitive Guide*. O’Reilly Media, Inc., 2002.

[102] “Long-running transaction,” *Wikipedia*. 09-May-2016.

[103] “REST and long-running jobs.” [Online]. Available: http://farazdagi.com/blog/2014/rest-long-running-jobs/. [Accessed: 09-Mar-2017].

[104] B. Kratz, “Protocols for long running business transactions,” *Tech. Rep. 17 Infolab Tech. Rep. Ser.*, 2004.

[105] N. Schuster, *Coordinating Service Compositions : Model and Infrastructure for Collaborative Creation of Electronic Documents*. KIT Scientific Publishing, 2013.

[106] “Web Services Coordination (WS-Coordination) Version 1.1 OASIS Standard incorporating Approved Errata.” [Online]. Available: https://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os/wstx-wscoor-1.1-spec-errata-os.html. [Accessed: 09-Mar-2017].

[107] K.-P. Mehdi, *Consumer Behavior, Organizational Development, and Electronic Commerce: Emerging Issues for Advancing Modern Socioeconomies: Emerging Issues for Advancing Modern Socioeconomies*. IGI Global, 2008.

[108] J. Postel and J. Reynolds, “File Transfer Protocol.” [Online]. Available: https://tools.ietf.org/html/rfc959. [Accessed: 21-Feb-2017].

[109] “Wget,” *Wikipedia*. 05-Jun-2017.

[110] “Remote backup service,” *Wikipedia*. 23-Dec-2016.

[111] “WebRTC Home | WebRTC.” [Online]. Available: https://webrtc.org/. [Accessed: 09-Mar-2017].

[112] S. Yuzhuo and H. Kun, “Design and realization of chatting tool based on web,” in *2013 3rd International Conference on Consumer Electronics, Communications and Networks*, 2013, pp. 225–228.

[113] “File Transfer Protocol,” *Wikipedia*. 17-Feb-2017.

[114] Nandokakimoto, “Design Patterns for Distributed Systems,” *Kakimoto Online*, 15-May-2008. .

[115] E. B. Fernandez, “The Authenticator Pattern,” 1999.

[116] “Service Design Patterns - Client-Service Interactions - Media Type Negotiation.” [Online]. Available: http://www.servicedesignpatterns.com/ClientServiceInteractions/MediaTypeNegotiation. [Accessed: 11-Mar-2017].

[117] “NGINX Load Balancing - HTTP and TCP Load Balancer,” *NGINX*. [Online]. Available: https://www.nginx.com/resources/admin-guide/load-balancer/. [Accessed: 20-Feb-2017].

[118] “How nginx processes a request.” [Online]. Available: http://nginx.org/en/docs/http/request_processing.html. [Accessed: 20-Feb-2017].

[119] R. Orfali, D. Harkey, and J. Edwards, *Instant CORBA*. New York: Wiley, 1997.

[120] “Service Design Patterns - Request and Response Management - Service Controller.” [Online]. Available: http://www.servicedesignpatterns.com/RequestAndResponseManagement/ServiceController. [Accessed: 11-Mar-2017].

[121] S. Paul, “Reliable Multicast Protocol (RMP),” in *Multicasting on the Internet and its Applications*, Springer US, 1998, pp. 229–243.

[122] “Chapter 2. Architecture.” [Online]. Available: https://docs.jboss.org/hibernate/search/4.1/reference/en-US/html/search-architecture.html. [Accessed: 20-Feb-2017].

[123] “Reliable Multicasting with the JGroups Toolkit.” [Online]. Available: http://www.jgroups.org/manual/html_single/. [Accessed: 20-Feb-2017].

[124] C. V. Ramamoorthy, “Computer Science and Engineering Education,” *IEEE Trans. Comput.*, vol. C-25, no. 12, pp. 1200–1206, Dec. 1976.

[125] "Wiley: Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches - Claudia Leopold." [Online]. Available: http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471358312.html. [Accessed: 24-May-2017].

[126] D. Howard, *Node.js for PHP Developers*. .

[127] "Node.js." [Online]. Available: https://nodejs.org/en/. [Accessed: 24-May-2017].

[128] "AWS Programs for Research and Education," *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/grants/. [Accessed: 24-May-2017].

[129] "Cloud Computing and Virtualization | Masters of Science in Software Engineering | SJSU." [Online]. Available: https://msse.sjsu.edu/cloud-computing-and-virtualization. [Accessed: 23-May-2017].

[130] "Advanced Computer Science (Cloud Computing and Big Data) - MSc - Canterbury," *The University of Kent*. [Online]. Available: /courses/postgraduate/1211/advanced-computer-science-cloud-computing-and-big-data. [Accessed: 23-May-2017].

[131] "Cloud Computing MSc - Postgraduate - Newcastle University." [Online]. Available: http://www.ncl.ac.uk/postgraduate/courses/degrees/cloud-computing-msc/#profile. [Accessed: 23-May-2017].

[132] "MSc in Cloud Computing | NCI." [Online]. Available: https://www.ncirl.ie/Courses/Course-Details/course/MSc-in-Cloud-Computing-MSCCLOUD5. [Accessed: 23-May-2017].

[133] "MSc Cloud Computing :: University of Essex." [Online]. Available: https://www.essex.ac.uk/courses/details.aspx?mastercourse=PG00484&subgroup=1. [Accessed: 23-May-2017].

[134] N. Leavitt, "Is Cloud Computing Really Ready for Prime Time?," *Computer*, vol. 42, no. 1, pp. 15–20, Jan. 2009.

[135] "Top 7 most common uses of cloud computing," *Cloud computing news*, 06-Feb-2014. [Online]. Available: https://www.ibm.com/blogs/cloud-computing/2014/02/top-7-most-common-uses-of-cloud-computing/. [Accessed: 17-May-2017].

[136] L.-J. Zhang, "EIC editorial: Introduction to the body of knowledge areas of services computing," *IEEE Trans. Serv. Comput.*, vol. 1, no. 2, pp. 62–74, 2008.

[137] H. P. Breivold and I. Crnkovic, “Cloud Computing education strategies,” in *2014 IEEE 27th Conference on Software Engineering Education and Training (CSEE T)*, 2014, pp. 29–38.

[138] X. Wang, G. C. Hembroff, A. B. Cerier, and B. W. Perrault, “Introducing Cloud Computing with a Senior Design Project in Undergraduate Education of Computer System and Network Administration,” in *Proceedings of the 2011 Conference on Information Technology Education*, New York, NY, USA, 2011, pp. 177–182.

[139] M. I. Tariq, S. Tayyaba, H. Rasheed, and M. W. Ashraf, “Factors influencing the Cloud Computing adoption in Higher Education Institutions of Punjab, Pakistan,” in *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, 2017, pp. 179–184.

[140] S. A. Mokhtar, A. Al-Sharafi, S. H. S. Ali, and A. Z. Al-Othmani, “Identifying the determinants of cloud computing adoption in higher education institutions,” in *2016 International Conference on Information and Communication Technology (ICICTM)*, 2016, pp. 115–119.

[141] S. Ashtari and A. Eydgahi, “Student Perceptions of Cloud Computing Effectiveness in Higher Education,” in *2015 IEEE 18th International Conference on Computational Science and Engineering*, 2015, pp. 184–191.

[142] Á. Mitchell and L. Cunningham, “Impact of cloud computing in Ireland’s institutes of higher education,” in *eChallenges e-2014 Conference Proceedings*, 2014, pp. 1–11.

[143] I. D. M. Ortega and J. A. B. Ricaurte, “Cloud Computing, trend importance and relevance for higher education,” in *2014 9th Computing Colombian Conference (9CCC)*, 2014, pp. 90–95.

[144] “Criteria for Accrediting Engineering Programs, 2016 – 2017 | ABET.” .

[145] “Cloud computing,” *Wikipedia*. 16-May-2017.

[146] “10 Hot Technology Trends For 2016,” *Network Computing*, 15-Jan-2016. [Online]. Available: http://www.networkcomputing.com/storage/10-hot-technology-trends-2016/520323524. [Accessed: 16-May-2017].

[147] “Home - IEEE Cloud Computing.” [Online]. Available: http://cloudcomputing.ieee.org/. [Accessed: 16-May-2017].

[148] M. C. Caraman, S. A. Moraru, S. Dan, and C. Grama, “Continuous Disaster Tolerance in the IaaS clouds,” in *2012 13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 2012, pp. 1226–1232.

[149] "Computing platform," *Wikipedia*. 12-May-2017.

[150] "Cloud Solutions - Amazon Web Services (AWS)," *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/solutions/. [Accessed: 16-May-2017].

[151] "Amazon Education." [Online]. Available: https://www.amazon.com/gp/feature.html?docId=1000412651. [Accessed: 08-Jun-2017].

[152] "Education Grants - Free Credits for University Computer Science Classes," *Google Cloud Platform*. [Online]. Available: https://cloud.google.com/edu/. [Accessed: 16-May-2017].

[153] "GitHub Student Developer Pack," *GitHub Education*. [Online]. Available: https://education.github.com/pack. [Accessed: 16-May-2017].

[154] jyoshii270000RYPJ, "IBM Cloud Education Wiki : IBM Cloud Education Resource Guide," 20-Oct-2009. [Online]. Available: https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/WebSphere%20Education%20Wiki/page/IBM%20Cloud%20Education%20Resource%20Guide. [Accessed: 16-May-2017].

[155] "Microsoft Azure in Education." [Online]. Available: https://azure.microsoft.com/en-us/community/education/. [Accessed: 16-May-2017].

[156] "Variables in Your Science Fair Project," *Science Buddies*. [Online]. Available: http://www.sciencebuddies.org/science-fair-projects/project_variables.shtml. [Accessed: 08-May-2017].

[157] "Dependent and independent variables," *Wikipedia*. 27-Apr-2017.

[158] P. R. Rosenbaum, *Design of Observational Studies*. New York, NY: Springer New York, 2010.

[159] "Observational study," *Wikipedia*. 02-Feb-2017.

[160] "Observational Studies." [Online]. Available: http://www.stat.wmich.edu/s160/book/node69.html. [Accessed: 04-May-2017].

[161] F. Shull, J. Carver, and G. H. Travassos, "An Empirical Methodology for Introducing Software Processes," in *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2001, pp. 288–296.

[162] "Observational Studies and Secondary Data Analyses To Assess Outcomes in Complementary and Integrative Health Care," *NCCIH*, 22-Jun-2012. [Online]. Available: https://nccih.nih.gov/research/blog/observational-secondary. [Accessed: 04-May-2017].

[163] A. Abou-Zeid and M. A. Taha, "Accreditation process for engineering programs in Saudi Arabia: Challenges and lessons learned," in *Global Engineering Education Conference (EDUCON), 2014 IEEE*, 2014, pp. 1118–1125.

[164] V. R. B.-G. Caldiera and H. D. Rombach, "Goal question metric paradigm," *Encycl. Softw. Eng.*, vol. 1, pp. 528–532, 1994.

[165] "Voyant Tools." [Online]. Available: https://voyant-tools.org/. [Accessed: 23-May-2017].

[166] "Analysis of variance," *Wikipedia*. 07-May-2017.

[167] "Interpret the key results for One-Way ANOVA." [Online]. Available: http://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/anova/how-to/one-way-anova/interpret-the-results/key-results/. [Accessed: 19-May-2017].

[168] "What a p-Value Tells You about Statistical Data," *dummies*. .

[169] "Sentiment analysis," *Wikipedia*. 10-May-2017.

APPENDICES

Appendix A: Pattern Language for Application-level Communication Protocols [10].

A Pattern Language for Application-level Communication Protocols

Jorge Edison Lascano[1,2], Stephen Wright Clyde[1]
[1]Computer Science Department, Utah State University, Logan, Utah, USA
[2]Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador
email: edison_lascano@yahoo.com, Stephen.Clyde@usu.edu

***Abstract*—Distributed applications depend on application-layer communication protocols to exchange data among processes and coordinate distributed operations, independent of underlying communication subsystems and lower level protocols. Since such protocols are application-specific, developers often must invent or re-invent solutions to reoccurring problems involving sending and receiving messages to meet specific functionality, efficiency, distribution, reliability, and security requirements. This paper introduces a pattern language, called CommDP, consisting of nine design patterns that can help developers understand existing reusable solutions and how those solutions might apply to their situations. Consistent with other pattern languages, the CommDP patterns are described in terms of the problems they address, their contexts, and solutions. The problems and consequences of the solutions are evaluated against four desirable qualities: reliability, synchronicity, longevity, and adaptability for scalable distribution.**

Keywords-design patterns; pattern languages; communication protocols.

I. Introduction

At the application level, a distributed system is two or more processes sharing resources and working together via network communications to accomplish a common goal [1][2]. Such systems are ubiquitous in today's Internet-connected world and are found in virtually every application domain, such as personal productivity tools, social media, entertainment, research, and business. Even single-user software systems that appear to be non-distributed may in fact communicate with other processes in the background to download updates, track usage statistics, or capture error logs, and are therefore actually distributed systems.

In general, the developers of a distributed system try to increase its overall throughput, reliability, and scalability by hosting data and/or operations on multiple machines, while minimizing network traffic, congestion, and turn-around times. Exactly how they do this depends heavily on the nature and requirements of the application. In some cases, developers may choose to distribute instances of one type of resource, e.g., image files in a peer-to-peer shared photo library. In other situations, developers may group resources such that all instances of a single type are on one server. Still in other cases, developers can take hybrid approaches, distributing certain types of resources among peers and hosting other types on dedicated servers. A closely related design issue deals with the granularity of the distributed resources, i.e., data and operations. From a data perspective, the possible choices range from whole databases to individual records or even individual fields within records. From an operations perspective, the choices range from entire subsystems to atomic operations. With today's programming languages, many developers follow the object-oriented paradigm, encapsulating operations with data and making choices for granularity that range from entire sets of objects to object fragments [3].

Besides deciding on the granularity and distribution of resources (data, operations, or objects), developers often have to consider requirements for security, fault tolerance, maintainability, openness, extensibility, scalability, and dynamic quality of service [2]. The degree to which an application possesses these desirable characteristics is primarily a consequence of architectural design choices, which, in turn, place new requirements on inter-process communications.

The problem is not that existing application-level communications protocols are poorly designed and implemented; rather, the problem is that application developer has to re-invent or re-design them for every new application.

In this paper, we will refer to an exchange among two or more processes for a particular purpose as a *conversation*. A single conversation may be short and simple, like querying a stock's price, or it could be long and complex, like the streaming of a video. The rules that govern a particular type of conversation are a *communication protocol* and a collection of protocols is called a *protocol suite* [4][5].

Application-layer communication protocols (ACPs) are often defined on top of other protocols. For example, the *Hypertext Transfer Protocol* (HTTP), which is an ACP, is defined on top of the *Transmission Control Protocol* (TCP) [1]. Many higher level ACPs, like webservice-based ACPs, are in turn defined on top of HTTP [1]. Section II provides additional background on protocols and protocol suites, as well as a brief discussion on layered communication subsystems.

Because requirements for ACPs can come from an application's (a) functional requirements, (b) architectural design, and (c) use of lower layer protocols, coming up with effective designs can be challenging. Fortunately, the problems that developers are likely to encounter are not uncommon and have known solutions. The key is to capture this knowledge in a way that developers can easily find it and adapt it to a new application. This is precisely what design patterns can do [6].

Unfortunately, design patterns for communication protocols at application layer have yet not been gathered, correlated, and formally organized into a cohesive and thorough collection. To this end, this paper introduces a system of design patterns, i.e., a pattern language, for ACPs,

Appendix B: Using Cloud Services To Improve Software Engineering Education for Distributed Application Development [24]

Using Cloud Services To Improve Software Engineering Education for Distributed Application Development

Jorge Edison Lascano[1,2], Stephen W. Clyde[1]
[1]Computer Science Department, Utah State University, Logan, Utah, USA
[2]Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador
email: edison_lascano@yahoo.com, Stephen.Clyde@usu.edu

***Abstract*—Software-engineering education should help students improve other development skills besides design and coding. These skills, referred to here as A2R (Analysis to Reuse), include analysis, technology evaluation, prototyping, testing, and reuse. The need for improved A2R skills is particularly pronounced in advanced areas like distributed application development. Hands-on programming assignments can be an important means for improving A2R skills, but only if they focus on the right details. This paper presents a case study of programming assignments offered in a graduate-level class on distributed application development, where the assignments required the students to use cloud services and programming tools that were heretofore unfamiliar to the students. Direct observation by the instructor and a post-class survey provided evidence that the assignments did in fact help students improve their A2R skills. The post-class survey also yielded some interesting insights about the potential impact of well-designed programming assignments, which in term led to ideas for future research.**

Keywords-computer science education; software-engineering education; cloud computing; virtual environments; distributed systems.

I. Introduction

Imagine yourself at a worktable with four or five of your peers. In the center of the table is a pile of seemingly random objects, including two dozen sheets of paper, paper clips, a small roll of tape, pins, and several small wooden sticks. A quick glance around the room reveals a dozen other groups just like yours with similar piles in front of them. An individual, who is introduced as your customer, stands at the front of the room and says that you have 30 minutes to build a "great" tower. What do you do first? How do you put all that you know about paper, clips, tape, wooden sticks, etc. into practice to satisfy the customer's request for a tower and do so within 30 minutes?

Such is the typical scene on the first day of class in the undergraduate introductory course on software engineering at Utah State University (USU). In general, all the students have a good working knowledge of objects at their disposal and even some inkling on how they may combine several of them to create new more structural useful objects. Most groups succeed in creating something that stands on its own and roughly resembles a tower within 30 minutes. However, at the end of that time, the customer surprises the students by giving them a few more objects, e.g., more paper and tape, and asks them to take 15 more minutes to make their towers taller or stronger. Many groups fail to do so in the limited allotted time. In fact, about half of them end up destroying their original towers in the attempt.

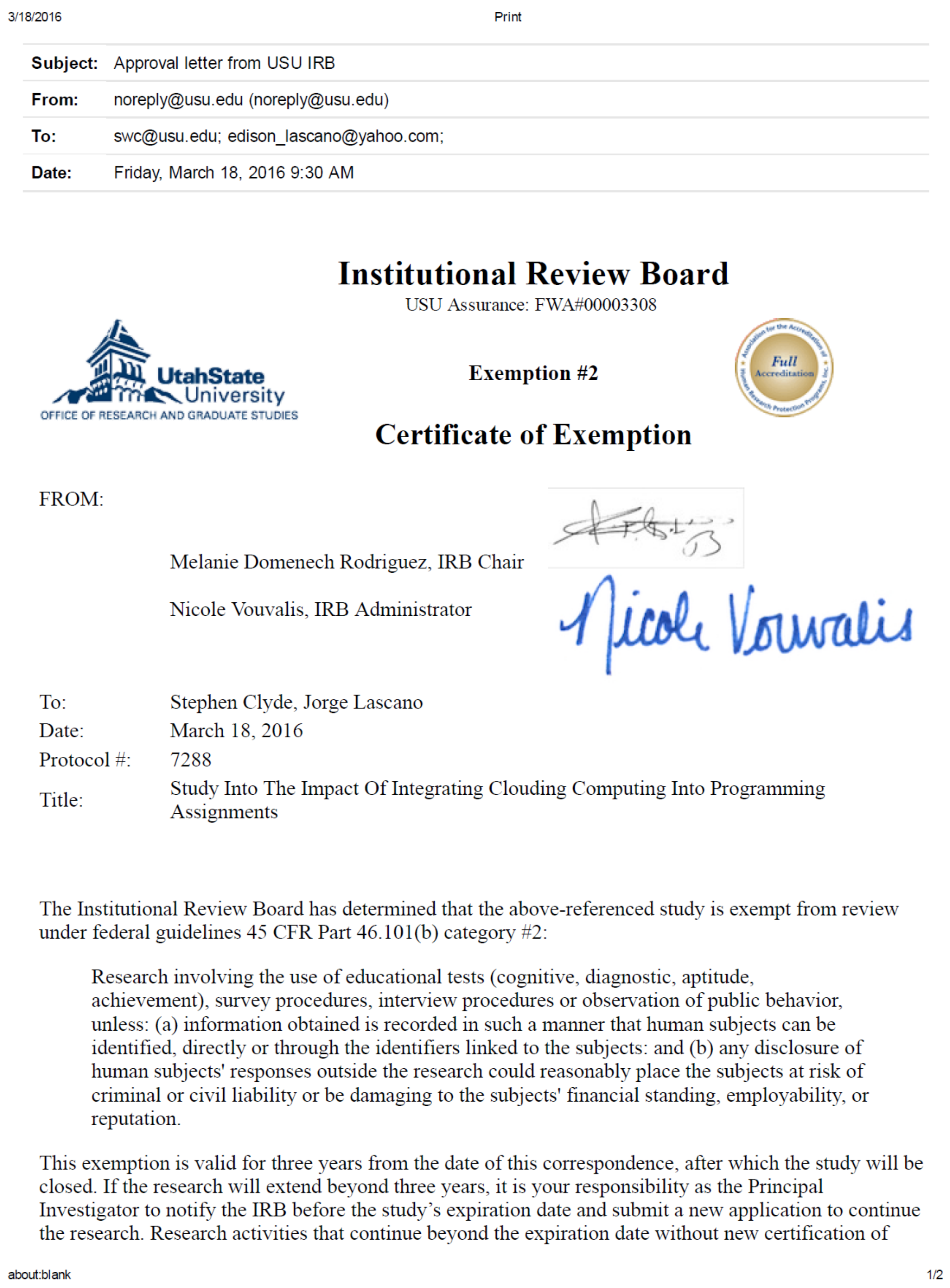
Afterwards the instructors and students discuss the experience in terms of what worked well for the group, particular difficulties that hindered progress, how the group organized itself, and how they decided on an overall approach. The discussion usually leads to some very interesting comparisons with common aspects of software engineering, such as group work, tool evaluation, prototyping, design patterns, testing, extensibility, reuse, and more. Over the years, one of the authors, who is a long-time instructor for this introductory software engineering course, has observed the following:

1. Virtually no student or group ever asks the customer what a "great" tower means. Most assume that they already know and proceed to build without each researching the requirements.
2. Virtually no student or group ever looks around to see what other groups have done or are doing, evaluates the ideas they see, and then tries to adapt or improve on them.
3. Only a small percentage of the groups try prototyping an idea to explore its characteristics.
4. Only rarely does a group test the properties (e.g., stability or strength) of a component or the whole tower and then try to make modifications to improve those properties.
5. Only a few groups try to establish patterns or "best practices" either in their building processes or the components they create, and then reuse those ideas.

Each of these observations represents a potential engineering pitfall or negative practice that can lead to inefficiency or failure. Software-engineering education needs to help students avoid these and other related pitfalls by connecting theory with best practices in the context of real non-trivial problems [1]. Doing so goes well beyond teaching the "How To's" of a specific technology, like a programming environment. Instead, it requires educators and students alike to address the "How To's" of the overall development process, including:

1. How do we know when we understand the customer's problem sufficiently?
2. How can we benefit from existing technology or from what others have tried in the past?
3. How can we prototype part of a problem or alternative solutions to answer critical questions?

Appendix C: IRB Approval.



3/18/2016 Print

Subject:	Approval letter from USU IRB
From:	noreply@usu.edu (noreply@usu.edu)
To:	swc@usu.edu; edison_lascano@yahoo.com;
Date:	Friday, March 18, 2016 9:30 AM

Institutional Review Board

USU Assurance: FWA#00003308

UtahState University
OFFICE OF RESEARCH AND GRADUATE STUDIES

Exemption #2

Association for the Accreditation of Human Research Protection Programs, Inc.
Full Accreditation

Certificate of Exemption

FROM:

Melanie Domenech Rodriguez, IRB Chair [illegible]

Nicole Vouvalis, IRB Administrator Nicole Vouvalis

To:	Stephen Clyde, Jorge Lascano
Date:	March 18, 2016
Protocol #:	7288
Title:	Study Into The Impact Of Integrating Clouding Computing Into Programming Assignments

The Institutional Review Board has determined that the above-referenced study is exempt from review under federal guidelines 45 CFR Part 46.101(b) category #2:

> Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless: (a) information obtained is recorded in such a manner that human subjects can be identified, directly or through the identifiers linked to the subjects: and (b) any disclosure of human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

This exemption is valid for three years from the date of this correspondence, after which the study will be closed. If the research will extend beyond three years, it is your responsibility as the Principal Investigator to notify the IRB before the study's expiration date and submit a new application to continue the research. Research activities that continue beyond the expiration date without new certification of

about:blank 1/2

exempt status will be in violation of those federal guidelines which permit the exempt status.

As part of the IRB's quality assurance procedures, this research may be randomly selected for continuing review during the three year period of exemption. If so, you will receive a request for completion of a Protocol Status Report during the month of the anniversary date of this certification.

In all cases, it is your responsibility to notify the IRB prior to making any changes to the study by submitting an Amendment/Modification request. This will document whether or not the study still meets the requirements for exempt status under federal regulations.

Upon receipt of this memo, you may begin your research. If you have questions, please call the IRB office at (435) 797-1821 or email to irb@usu.edu.

The IRB wishes you success with your research.

4460 Old Main Hill Logan, UT 84322-4460 PH: (435) 797-1821 Fax: (435) 797-3769 WEB: irb.usu.edu EMAIL: irb@usu.edu

Appendix D: Survey for CS6200/CS7930 to Measure the Perception of Increased/Decreased Knowledge and Skills on a Course Using CC for the Assignments

Q1 For each CONCEPT listed below, rate your knowledge level on a scale of 1 to 5 as it was BEFORE you took the CS6200/7930 -- 1=Low and 5=High.

- Inter-process communications patterns, like Request-Reply, Request-Reply Acknowledge, and Reliable Multicasts
- Partial ordering of events in a distributed system, as represented by mechanism like Vector Timestamps
- Message serialization/deserialization
- Intra-process concurrency
- Inter-process concurrency
- Computation resources in a cloud-computing environment, such as AWS
- Namespaces, name services, and name resolution
- Deployment, execution and testing techniques in a distributed environment
- Deployment, execution and testing techniques in the cloud.
- Distributed election algorithms
- Resource managers
- Fault tolerance in a distributed environment.
- Tools for provisioning collection of resources needed for a distributed system.
- Cloud Computing resources
- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)

Q2 For each SKILL listed below, rate your knowledge level on a scale of 1 to 5 as it was BEFORE you took the CS6200/7930 -- 1=Low and 5=High.

- AWS Users and key pairs (Identity and Access management -IAM)
- AWS Virtual PC Instances (EC2)
- AWS Storage (S3, EBS)
- AWS-CLI (Command Line Interface)
- AWS SDK (Software Development Kit)
- Managing instances in AWS: creating/launching, starting, stopping, terminating
- AWS Billing
- Using Node.js to Develop Distributed Systems
- Using Node.js to deploy and run Distributed Systems in the cloud
- Designing and developing TCP/UDP/Web Services-based systems with Node.js
- Writing scripts to Deploy/execute applications in distributed environments
- Designing and Developing Resource Managers
- Designing and Developing Distributed Election Algorithms

Q3 For each CONCEPT listed below, rate your knowledge level on a scale of 1 to 5 AFTER you took the CS6200/7930 -- 1=Low and 5=High.

- Inter-process communications patterns, like Request-Reply, Request-Reply Acknowledge, and Reliable Multicasts
- Partial ordering of events in a distributed system, as represented by mechanisms like Vector Timestamps
- Message serialization/deserialization.
- Intra-process Concurrency
- Inter-process Concurrency
- Computation resources in a cloud-computing environment, such as AWS
- Namespaces, name services and name resolution
- Deployment, execution and testing techniques in a distributed environment
- Deployment, execution and testing techniques in the cloud.
- Distributed election algorithms
- Resource managers
- Fault tolerance in a distributed environment.
- Tools for provisioning collection of resources needed for a distributed system.
- Cloud Computing resources
- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)

Q4 For each SKILL listed below, rate your knowledge level on a scale of 1 to 5 AFTER you took the CS6200/7930 -- 1=Low and 5=High.

- AWS Users and key pairs (Identity and Access management -IAM)
- AWS Virtual PC Instances (EC2)
- AWS Storage (S3, EBS)
- AWS-CLI (Command Line Interface)
- AWS SDK (Software Development Kit)
- Managing instances in AWS: creating/launching, starting, stopping, terminating
- AWS Billing
- Using Node.js to Develop Distributed Systems
- Using Node.js to deploy and run Distributed Systems in the cloud
- Designing and developing TCP/UDP/Web Services-based systems with Node.js
- Writing scripts to Deploy/execute applications in distributed environments
- Designing and Developing Resource Managers
- Designing and Developing Distributed Election Algorithms

Q5 For each of the following activity, indicate if you are in favor of using AWS (or other some cloud environment) for that activity.

- ❑ Testing the behavior of applications in a distributed environment (e.g., HW2)
- ❑ Understanding of distributed systems concepts such as serialization, concurrency, shared resources, etc. (e.g., HW4)
- ❑ Programming in other courses. (If yes, what courses would you advise)

____

Q6 What general lessons learned in Software Engineering did you learn by using AWS?

Q7 What lessons in Distributed System (other than those address in above) did you learn by using

AWS?
Q8 What lessons in Testing and Debugging (other than those address in above) did you learn by using AWS?
Q9 What lessons in Deployment (other than those address in above) did you learn by using AWS?
Q10 What lessons in Fault Tolerance (other than those address in above) did you learn by using WS?

Appendix E: Survey to Measure Perception of Increased/Decreased Knowledge/Skills for Fall 2015 Semester, Same is Applied on Spring 2016 and Fall 2016 Semesters.

USU CS students skills FALL-2015
Q. a. SCROLL DOWN TO START THE SURVEY and GET ONE OF THE FOUR $25.00 GITF CARDS
Introduction/ Purpose. You have been asked to take part of this survey because you were/are enrolled in one or more CS courses that can be related to the use of Cloud Computing Resources to improve software engineering education. There will be approximately 129 total participants in this research.
Procedures. If you agree to be in this research activity, you will be asked to fill out a survey to measure your perception of skills and knowledge acquired from computer science classes. This survey will be distributed at the beginning of Spring 2016, Fall 2016, and Spring 2017 semesters. Please participate at least once. Every survey will take approximately between 10 to 15 minutes.
Risks Participation in this research activity may involve some risks or discomforts. These include a minimal risk of loss of confidentiality, we will take steps to reduce the risk. The information obtained during this research activity is intended to contribute to generalizable knowledge and as such it is intended for publication. Whether you decide to participate or not will not affect your standing with the CS program.
Benefits. Your participation in this research is not expected to lead to any direct benefit to you. Researchers hope that the result from this survey will help determine whether integrating the use of cloud computing in course assignments can help the CS department meet Program Educational Objectives as outlined by the department's accrediting body. You may derive personal satisfaction from supporting a student in acquiring important professional skills. Researchers will learn about the analysis of skills and knowledge to measure the improvement of Software Engineering Education methods that use Cloud Computing Resources.
Explanation & offer to answer questions. Jorge Edison Lascano has explained this research activity to you and answered your questions. If you have other questions or research-related problems, you may reach the student researcher, Jorge Edison Lascano at (435) 213-5529 or edison_lascano@yahoo.com and the Principal Investigator Dr. Stephen W. Clyde at (435) 764-1596 or stephen.clyde@usu.edu
Payment/Compensation. There is no compensation for your participation in this research activity. There will be a drawing of four $25.00-gift-card among the survey respondents.
Voluntary nature of participation and right to withdraw without consequence. Participation in this research activity is entirely voluntary. You may refuse to participate or withdraw at any time without consequence. You may be withdrawn from this study without your consent by the investigator if the information may be considered wrong, for example the course IDs provided by the students.
Confidentiality Records from this research activity will be kept confidential, consistent with federal and state regulations that apply to research. Only Jorge Edison Lascano and Stephen Clyde will have access to the data which will be kept in a locked file cabinet or on an encrypted USU computer account to maintain confidentiality. To protect your privacy, personal, identifiable information, such as your A number, will be removed from study documents and replaced with a study identifier. Identifying information will be stored separately from data and will be kept for the duration of the study, approximately March 2016 to June 2017. All information will be

destroyed by June 2017.
IRB Approval. Statement The Institutional Review Board for the protection of human participants at Utah State University has approved this research study. If you have any questions or concerns about your rights or a research-related injury and would like to contact someone other than the research team, you may contact the IRB Director at (435) 797-0567 or irb@usu.edu to obtain information or to offer input.
Student Statement. “I certify that the research activity has been explained to the individual, by me, and that the individual understands the nature and purpose, the possible risks and benefits associated with taking part in this research study. Any questions that have been raised have been answered.”
Researchers:

Stephen W. Clyde, PhD	Jorge Edison Lascano
Principal Investigator	Student Researcher
Telephone—435-764-1596	Telephone—435-213-5529
stephen.clyde@usu.edu	edison_lascano@yahoo.com

Thank you for your participation.

Q. b. Click in the arrow below to start.
Q. c. Do you agree to participate in this survey
- ❍ Yes
- ❍ No

If No Is Selected, Then Skip To End of Survey

Q0 What is your Age?
- ❍ under 18 years old
- ❍ 18-65 years old
- ❍ 65 years or older

If under 18 years old Is Selected, Then Skip To End of Survey If 18-65 years old Is Not Selected, Then Skip To End of Survey

Q1 Did you take any CS courses during Fall 2015 semester?
- ❍ Yes
- ❍ No

If No Is Selected, Then Skip To End of Survey

Q2 What is your A#?A01234567

Did you take any CS courses during Fall 2015 semester? Yes Is Selected

Q3 What CS course(s) did you take in Fall 2015. Enter the course numbers of those classes (up to five).

CS

…

Q4 CS courses enrolled for Spring 2016. Enter the course numbers of those classes (up to five)

CS

…

Q5 For each SKILL listed below, assess your level of expertise on a scale of 1 to 5, as you remember them being at the beginning and end of Fall 2015. 1=Low and 5=High, with n/a meaning you have no experience with that skill.

	BEGINNING of the Fall 2015 semester						END of the Fall 2015 semester					
	5	4	3	2	1	n/a	5	4	3	2	1	n/a
(a) Ability to analyze real world software needs or requirements	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(b) Ability to evaluate methods, tools, techniques, libraries, or components for re-use	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(c) Ability to reuse methods, designs, or software from previous assignments	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(d) Ability to design a software system to meet complex real-world requirements	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(e) Ability to implement a software system according to a design	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(f) Ability to implement software that can run in a runtime environment different from your own computer or one in a school lab	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(g) How to configure a runtime environment into which you can deploy a software system that you built	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(h) Ability to deploy a system to a runtime environment different from your own computer or one in a school lab	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(i) Ability to learn and use virtual or cloud-based resources for creating software solutions	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(j) Ability to thoroughly test software systems using executable test cases	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(k) Ability to think critically and develop alternative solutions to a problem	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍

(l) Ability to think creatively about software solutions	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(m) Ability to understand, evaluate and use emerging technologies	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(n) Ability to learn and use existing software services available on the Internet	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(o) Ability to follow industry-wide "best practices" when use your chosen development environment	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(p) Ability to follow "best practices" in testing	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(q) Ability to follow "best practices" in deploying software to runtime environments	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(r) Ability to apply green practices	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(s) Ability to evaluate a variety of operating systems and frameworks as possible runtime environments	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(t) Ability to apply collaborative methods, tools, techniques to develop software	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(u) Ability to design a software system that will provide for a good user experience	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(v) The ability to create maintainable software	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(w) Ability to understand real-world problems related to the course material	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍

Q6 For each KNOWLEDGE listed below, assess your level of understanding on a scale of 1 to 5, as you remember them being at the beginning and end of Fall 2015. 1=Low and 5=High, with n/a meaning you have no understanding of that knowledge.

	BEGINNING of the Fall 2015 semester						END of the Fall 2015 semester					
	5	4	3	2	1	n/a	5	4	3	2	1	n/a
(a) Understanding of how to match the needs of an application to an appropriate develop environment and runtime platform	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(b) Understanding of at least one development stack (a collection of reusable components or libraries)	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(c) Understanding of development environment setup	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(d) Understanding of principles of software testing	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(e) Understanding of network communications	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(f) Understanding of the principle of reliability as it applies to software systems	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(g) Understanding of security principles and practices for software systems	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(h) Understanding of what affect runtime performance and how to detect inefficiencies and correct them	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(i) Understanding of the principle of Coupling	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(j) Understanding of the principle of Cohesion	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(k) Understanding of software reuse	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(l) Understanding "abstraction" with respect to the design and implementation of software system.	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(m) Understanding "encapsulation" with respect to the design and implementation of software systems	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍
(n) Understanding "modularization" with respect to the design and implementation of a software system	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍

Q7 For each QUESTION listed below, express your level of agreement on a scale of 1 to 5 (5=Completely Agree and 1=Low agreement), as you remember it being at the end of the Fall 2015 semester

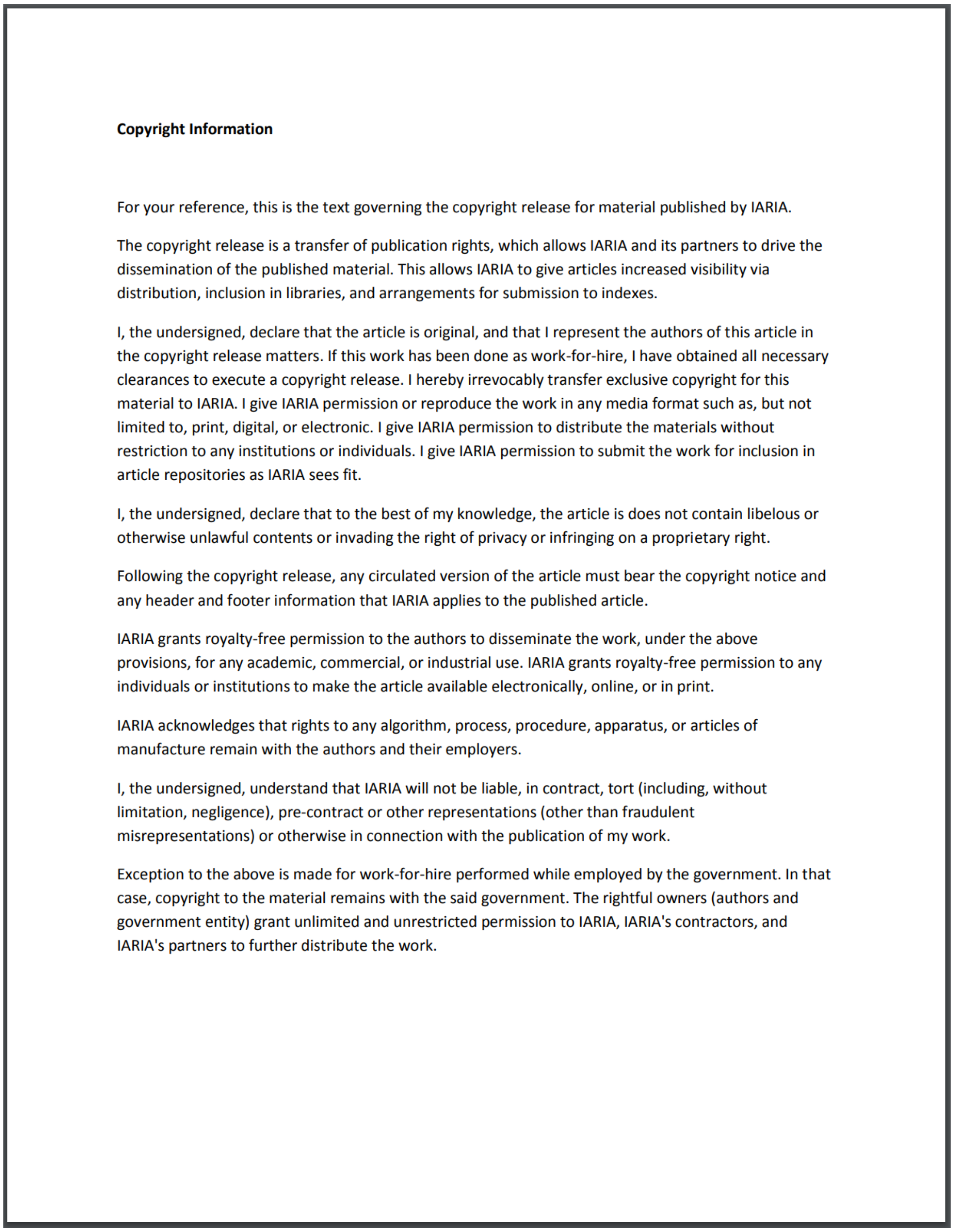
	1	2	3	4	5
(a) Assignments design comply with realistic execution environments characteristics?	❍	❍	❍	❍	❍
(b) Student's possibility to apply course learnings in real-world examples	❍	❍	❍	❍	❍
(c) The design principles are important for CS courses?	❍	❍	❍	❍	❍

Q8. How do you think that your skills to design, implement and evaluate software systems can be best improved?
Q9. What methods, tools, or techniques would you recommend to courses use for course assignment that would help you adapt to emerging technologies?
Q10. We will use your A number for the drawing of four $25.00 Gift Cards. The A number you have entered is A…. If this value is wrong, you can return to the first question to correct it. Do you want to participate in the drawing?

❍ Yes A…

❍ No

Appendix F: Copyright Page from ICSEA Proceedings by IARIA



Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

CURRICULUM VITAE

Jorge Edison Lascano

CAREER OBJECTIVE:

To obtain a position in the academia, especially in my country Ecuador, where Higher Education requires high level researchers and instructors to contribute with the future professionals learning in the Computer Science fields.

EDUCATION:

PhD in Computer Science, Utah State University, Logan, Utah. Grad GPA 4.0. MS in Software Engineering, San Jose State University, San Jose, California. Certificate in Higher Education Management, Universidad de Las Fuerzas Armadas -ESPE, Sangolquí, Ecuador. BS in Informatics and Computer Systems Engineering, Escuela Politécnica Nacional, Quito, Ecuador, Honors Curriculum, Graduate Cum Laude.

DEGREES AND HONORS:

Fulbright Docent Scholarship recipient, July 2007 - July 2009

PROFESSIONAL EXPERIENCE:

Technical Intern III, Yahoo INC!, 2008. Research Assistant, Utah State University, 2012-2014. Technical Support, Smart Choice Communications, 2005-2007. Developer, Ecuadorian Disabilities National Council, 2000-2001. Software Engineer, Pichincha Sistemas, 1999. Lab Auxiliar, Escuela Politécnica Nacional, 1995-1998

TEACHING EXPERIENCE:

Associate Professor, Universidad de las Fuerzas Armadas ESPE, Ecuador, 2003-2011

Faculty member, Escuela Politécnica Nacional, Ecuador, 1998-2007, 2009-2011

Faculty member, Universidad de las Américas, Ecuador, 2005-2007, 2009-2011

Faculty member, UNIANDES University, Ecuador, 2005-2006

Faculty member, Universidad Técnica de Ambato, Ecuador, 2001

Graduate Instructor, Utah State University, 2013

Graduate Teaching Assistant, Utah State University, 2014-2017

PUBLICATIONS:

1. J. E. Lascano and S. Clyde, "A Pattern Language for Application-level Communication Protocols," presented at the ICSEA 2016, The Eleventh International Conference on Software Engineering Advances, 2016, pp. 22–30.
2. J. E. Lascano and S. W. Clyde, "Using Cloud Services To Improve Software Engineering Education for Distributed Application Development," presented at the ICSEA 2015, The Tenth International Conference on Software Engineering Advances, 2015, pp. 438–444.
3. Raza, J. E. Lascano, and S. Clyde, "Communication Aspects with CommJ: Initial Experiment Show Promising Improvements in Reusability and Maintainability," presented at the ICSEA 2014, The Ninth International Conference on Software Engineering Advances, 2014, pp. 48–54.
4. J. E. Lascano, "eXtreme Programming (XP) May Be Embedded Inside Scrum," in 21st Annual Systems and Software Technology Conference (STSC) 2009: Technology: Advancing Precision, 2009.