PRINCIPLES AND PROPERTIES FOR REDUCING THE PREVALENCE OF IMPLICIT INTERACTIONS IN SYSTEM DESIGNS

BY

LUKE NEWTON

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Applied Science
in
Electrical and Computer Engineering

Carleton University
Ottawa, Ontario, Canada

Abstract

Early security considerations are essential to ensuring a system is adequately protected, but their ever-growing size and complexity often leaves full comprehension of a system's interconnections out of reach. This gives rise to implicit interactions. These unplanned or unforeseen communication sequences between components are security vulnerabilities that can be exploited to mount a cyberattack. Existing design-phase formal methods-based approaches exist to identify implicit interactions, but formal methods see limited adoption and the root cause of implicit interactions is not well understood.

In this work, we extend the existing formal approach to suggest areas of a system to focus redesign efforts, while also providing alternative approaches that do not require formal expertise. These focus on graph-based measurements and providing a set of properties, quality attributes, and design principles with goals in line with the reduction of the prevalence of implicit interactions within a system design.

Acknowledgements

I would like to thank my supervisor, Professor Jason Jaskolka, for his interest and guidance in this project. His support, encouragement, and ability to foster ideas and out-of-the-box thinking were all pivotal to the success of this work.

I thank my colleagues for creating an inviting environment to share ideas, and for all their advice and constructive comments.

Finally, I wish to thank my parents and my best friend for all their unconditional love and support over these past two years.

Contents

List of Tables

List of Figures

List of Abbreviations

C^{2}KA Communicating Concurrent Kleene Algebra

CKA Concurrent Kleene Algebra

ICS Industrial Control Systems

IT Information Technology

MCCS Manufacturing Cell Control System

OT Operational Technology

WDS Wastewater Dechlorination System

Chapter 1

Introduction

In 2017, an American casino's high-roller database was hacked through a smart thermometer in the casino's fish tank [Mat17]. The response to such an incident raises many questions, one such question being: why can a fish tank thermometer communicate with the high-roller database at all? Internet-of-Things-enabled smart devices such as a thermometer often lack adequate security [KKSV17], but in order to carry out an attack with the compromised device, it must have some potential for communication with the target. Earlier, in 2015, security researchers showed how by compromising the infotainment system on a Jeep, they were able to take control over the vehicle's engine, brakes, and transmission, leading to a recall of 1.4 million vehicles [Gre15]. Again, the question is: why should a vehicle's infotainment system be able to communicate with, or have any influence over, critical systems such as the transmission or brakes? Regardless of how the thermometer or infotainment system

are compromised, the design of such systems can be used to improve security by disallowing these system components that have no business talking with one another to interact, preventing such attacks from ever occurring.

The difficulty is that these types of interactions between components, if unintended, may not be known to designers of the system. Such interactions, known as implicit interactions, are the focus of this work. To begin, we introduce the concept of implicit interactions, the threat they pose, and the motivation for this work. Section 1.1 introduces some basic definitions used throughout this work and defines the scope of the thesis. Section 1.2 broadly describes the motivations behind addressing the problem and research questions described in Section 1.3. Section 1.4 summarizes the main contributions of the thesis. Finally, Section 1.6 describes the structure of the remainder of the thesis.

1.1 Context

As the systems we rely upon continue to grow in size and complexity, security becomes an increasingly desired system trait. Unfortunately, security is often only retrofitted or "bolted-on" to many of the systems that we build [Jas20c]. This can lead to systems with inadequate security solutions, security solutions that feel disjointed or conflicted from the rest of the system architecture, and higher maintenance costs due to increased amounts of patching required to close holes in the system's defenses.

Security-related activities should be included in each phase of a project's lifecycle for improved security solutions [RMO16], but considering security and possible vulnerabilities earlier is particularly beneficial. Evaluating a system's design with respect to its cybersecurity vulnerabilities provides an opportunity to resolve potential issues before they can be exploited by attackers, and at earlier stages it is generally less expensive to implement any required changes [DBRB10]. In systems such as critical infrastructure, it is important we have adequate security and that everything works the first time around, since any successful attacks can have catastrophic impacts.

Industrial Control Systems (ICS) compose much of today's critical infrastructure, and are of particular interest. ICS are a category of systems that control and automate the stable operation of industrial processes in a wide variety of settings, including power generation, manufacturing, and chemical production [MKW+16]. Nowadays, ICS fall not only into the category of Operational Technology (OT), but increasingly into the category of Information Technology (IT) due to the continued integration of cyber and physical components, as well as connections between the system which manages the physical industrial process and the company's wider IT network. ICS are subject to a wide range of attacks [MKW+16, GKA17, Mic, ZJS11], and the task of ensuring adequate security for ICS has become considerably more challenging as IT and OT considerations can, at times, be in opposition to each other [MJV17].

In this work, a system is considered as a set of interacting agents that communicate with each other to achieve a set of system objectives. Here, an *agent* refers to any component whose behavior consists of discrete actions, and where each interaction of an agent with its neighbouring agents is called a *communication* [Mil89]. Figure 1.1

Figure 1.1: A Simple Example System

shows an abstract system in which the boxes labelled A, B, and C are agents, and the arrows show communications.

In this view, system functionality is represented as intended sequences of communication between agents to coordinate their behaviors [JV17b]. We call these sequences of communication the system's *intended interactions*. However, there also exists the possibility for *implicit interactions* to exist in the system, which refer to any potential for communication [JK14] that is unfamiliar, unplanned, or unexpected, and is either not visible or not immediately comprehensible by the system designers [JV17b]. For example, suppose in Figure 1.1 that the system's objectives are achieved through A communicating with B, and C communicating with B. If there exists the potential for A to invoke a state change in B that ultimately influences C, or vice versa, then there are implicit interactions in the system. In this example the implicit interaction is quite easy to see, but in more realistic systems with dozens, hundreds, or thousands of agents and communications, it becomes impossible for any one practitioner to fully comprehend all possible communication sequences.

1.2 Motivation

Implicit interactions indicate the presence of a cybersecurity vulnerability that can be exploited to mount a cyberattack. By compromising the agent at the beginning of

an implicit interaction's communication sequence (the *source agent*), an attacker can influence the behavior of the agent at the end of the communication sequence (the *sink agent*) to perform actions other than the functions the system was intended to perform, impacting the system's stability, safety, and security [Jas20a]. The impacts of such an attack are of particular concern in ICS, where agent behaviors may correspond to actions performed by physical components, and so implicit interactions pose a threat not just to software system integrity, but also to the integrity of physical components, the process controlled by the ICS, and the safety of individuals around the system.

An implicit interaction can be exploited to compromise any of confidentiality, integrity, or availability – the so-called "CIA triad" of security [SB18]. An indirect, hidden potential for communication between agents in a system could be used to leak confidential information [JK14], such as in the case of the 2017 casino database hack. System or data integrity can be compromised in the sink agent, as well as those intermediate agents along the communication sequence as the conditions set up to influence the sink agent are possibly different from those that the system would experience in its normal operations. This is the case in the 2015 Jeep recall. System availability can be impacted if the compromised system or data integrity results in a deadlock.

Previous work has developed a rigorous, formal methods-based approach that can be used to identify the existence of implicit interactions in critical infrastructures and industrial control systems [JV17b]. This approach involves the specification and analysis of the communication among system components using the Communicating

Concurrent Kleene Algebra (C^2KA) modelling framework [Jas15, JKZ14]. This formal analysis provides the set of implicit interactions that exist in the system, but this information alone does not provide a clear path forward to remediate these vulnerabilities, nor does it point towards the root cause of the vulnerabilities, without further analysis. This crucial next step in the iterative design process was highlighted in recent work which applied the implicit interaction identification method to a real-world wastewater dechlorination system, in which operators of the system who aided in the system's modelling, stated that "a summary of problematic areas would be helpful as part of the reporting of the results" and that "if used in the early stages of system development, the approach can identify hidden problems and perhaps provide cost savings and time" [Jas20b]. The ability to identify "problematic areas" within a system would allow practitioners' limited time and resources to be properly focused, so we can most effectively mitigate implicit interactions given the available resources.

Furthermore, while formal methods provide high assurance levels for security and are sometimes required for systems that need high these high levels of confidence [CGD+16, Man03, AGM+19, Com17], there are scalability issues that make modelling large systems and wider adoption challenging [CGD+16, RCF+20], including the need to repeat the formal analysis from the beginning when a design change is made. If we can determine the root causes of implicit interactions and we can translate these to directly measurable quantities from the system model, then we can use these measurements as an indicator of the state of the system's implicit interactions as design changes are made. A full formal analysis can still be performed at the end of the design phase to provide evidence for security assurance, or at any point throughout the design process, but with simpler measurements that correlate highly with the

prevalence of implicit interactions, a formal analysis would not always necessary to obtain useful information about this system view.

1.3 Problem Statement

Due to the increasing size, complexity, and interconnectedness of ICS, the potential for communication between agents in a system becomes more difficult to fully grasp, and gives rise to implicit interactions that can be exploited to launch a cyberattack. There are existing methods to identify implicit interactions, but how to mitigate this threat is not well understood.

Additionally, because implicit interactions are based on the *potential* for communication between agents, they are difficult to fully eliminate from a system. Removing all communication between agents would achieve this goal, but some amount of communication between agents is required for the system to perform its functionalities. Any design change to eliminate specific implicit interactions may also introduce new implicit interactions into the system, and making a specific design change to eliminate implicit interactions can preclude other changes that target different implicit interactions. A method to determine "problematic areas" within the system can also help with this issue, as it can help to ensure we make a more informed decisions backed by data, which will more effectively mitigate the implicit interactions in a system. Ultimately, we aim to reduce the *prevalence of implicit interactions* in a system, meaning we want to reduce the degree to which these vulnerabilities exist across the whole

system to minimize the number of areas from which an attack can be mounted (i.e., we aim to reduce the attack surface of a system).

We would also like to know how implicit interactions fit into the larger task of system design. Any design change made to mitigate implicit interactions is likely to have side effects on the rest of the system. When designing with a focus on implicit interactions, we want to be aware of how new design solutions impact other system goals, such as a system's target quality attributes. Design work is often guided by principles which encourage best practices, and an understanding of how these design principles capture the notion of implicit interactions, or whether additional considerations are needed, would be beneficial to understanding implicit interactions as a whole.

To fill this knowledge gap, this work aims to address the following research questions:

RQ1. How can we identity areas of a system design that should be modified to best reduce the prevalence of implicit interactions within the system?

RQ2. What measurable aspects of a system design are related to the prevalence of implicit interactions in that system?

RQ3. How do design changes that mitigate implicit interactions relate to system properties and system quality attributes?

RQ4. What existing design principles align with the goal of reducing the prevalence of implicit interactions in a system design?

1.4 Contributions

The main contributions of this work are as follows:

C1. **An approach for identifying weak points in a system design with respect to implicit interactions.** In Section 5.2, we present a systematic approach to identify problematic aspects of a system's design by analyzing the implicit interactions present in the design. The proposed approach can be used to prioritize re-design efforts and focus where the system designer's time and effort can be most effectively spent to mitigate implicit interactions. (i.e., we answer **RQ1**.).

C2. **Identification of measurements that relate to the prevalence of implicit interactions in a system.** In Section 5.4, we compare the results of existing formal methods and **C1**. with a graph-based analysis to identify graph-based approximations for selecting design alternatives with a lower prevalence of implicit interactions. This makes improving security in the design phase of a project more accessible beyond industry practitioners with formal expertise, as graph-based data collected from a system design can be used to understand whether the prevalence of implicit interactions in alternative system designs is improved or worsened, and can forecast which parts of a system are most vulnerable to cyberattacks that exploit implicit interactions. (i.e., we answer **RQ2**.).

C3. **An evaluation of how implicit interactions relate to system properties, quality attributes, and design principles.** By combining implicit

interaction and graph-based data, in Chapter 7 we uncover what phenomena in designs coincide with a higher prevalence of implicit interactions within a system and map these to design principles that serve as a another practical way to reduce the prevalence of implicit interactions without requiring formal expertise. (i.e., we answer **RQ3.** and **RQ4.**).

1.5 Related Publications

The following is a list of publications related to the work presented in this thesis.

- Luke Newton and Jason Jaskolka. Analyzing implicit interactions to identify weak points in cyber-physical system designs. In *2021 Resilience Week (RWS)*, pages 1–8. IEEE, 2021.

1.6 Structure

The remainder of this thesis is structured as follows:

Chapter 2 provides a more detailed description of implicit interactions and reviews the state-of-the-art in applying of formal methods, measurements, and design principles in the context of security.

Chapter 3 provides an overview of the activities involved in the proposed approach to answer our research questions and the required system information.

Chapter 4 introduces the modelling framework used in this work, C^2KA [Jas15] and the illustrative examples used in subsequent chapters.

Chapter 5 details how we identify implicit interactions, extend the existing formal methods approach to answer research question **RQ1.**, and take several graph-based measurements from design alternatives of each illustrative example to build a dataset.

Chapter 6 analyzes the dataset constructed in Chapter 5 to identify relationships between implicit interactions and the graph-based measurements to answer research question **RQ2.**.

Chapter 7 interprets the analysis and draws from the literature to answer research questions **RQ3.** and **RQ4.**.

Chapter 8 assesses the strengths and limitations of the proposed approach.

Chapter 9 provides concluding remarks and briefly discusses some possible future works.

Chapter 2

Literature Review

In this chapter, we review the state-of-the art for our problem domain. Section 2.1, introduces relevant work on implicit interactions, Section 2.2, reviews a wide variety of approaches to characterize security in systems, and Section 2.3 describes in detail how graph-based measurements have been used to improve system security and other qualities. Section 2.4 reviews the motivation behind the creation and adoption of design principles. Each section ends with a discussion to identify the gap in research that this work aims to fill.

2.1 Approaches to Identify and Analyze Implicit Interactions

Implicit interactions are a class of system vulnerability in which there is a potential for communication between two agents in the system that is not part of the intended system interactions [JV17b]. Regardless of the factors that result in the creation of an implicit interaction, communication pathways that are not part of the intended system interactions offer additional means by which one agent can influence another. As these pathways are, by definition, implicit and unintended, they are more likely to go unnoticed by designers and testers, and remain hidden in the system until exploited a malicious actor.

Implicit interactions are exploited through a *compromised agent*, which refers to an agent that can behave in a way that is not consistent with its original or intended specification. This means that it has the ability to issue any stimulus and/or alter its concrete behavior (e.g., by defining a shared environment) [Jas20a]. The compromised agent then, can exploit the implicit interactions in the system for which it serves as the source agent (e.g., by issuing stimuli or defining shared environments out-of-sequence from the intended system interactions) to cause knock-on affects that adversely impact other agents in the system. Implicit interactions can be identified with existing methods and tool support [Jas15, JV17b], which require a formal model of the system using the Communicating Concurrent Kleene Algebra (C^2KA) framework [JKZ14, Jas15].

A metric called *severity* has been developed to provide an indication of how "hidden" or unexpected an implicit interaction is in the system [JV17a]. The approach is meant to give an idea of how much overlap exists between any two communication paths. The less overlap between an interaction path and the intended interactions, the more unexpected that interaction is, and so it may pose a higher threat to the system it which it exists.

While severity is concerned with the threat posed by an implicit interaction, *exploitability* has been proposed to measure how many actions are available to a source agent that are able to influence the sink of an implicit interaction interaction [Jas20a]. A lower exploitability value means that each agent in an implicit interaction has fewer possibilities to cause a chain reaction that ultimately propagates influence all the way through each agents' neighboring agents.

Both severity and exploitability have been demonstrated on a real-world wastewater dechlorination system [Jas20b]. While other works related to implicit interactions use hypothetical systems for illustrative purposes [JK14, JV17b, JV17a, Jas20a], operators at an existing municipal wastewater dechlorination system worked closely with the author of [Jas20b] to create and validate a C^2KA model of the system the operators maintain. This work, in particular feedback from the operators on the analysis results, highlighted the applicability and strengths of identifying implicit interactions, as well as the need for a methodology to use the identified interactions and/or their measured properties to locate areas of improvement to reduce the prevalence of implicit interactions within a system.

With existing methods to identify and analyze implicit interactions, the next logical step is to explore how we can remediate these vulnerabilities. This work aims to directly address the need highlighted by the operators of the wastewater dechlorination system as one way to explore remediation options, as well as explore what existing design principles help to reduce the prevalence of implicit interactions within a system through their application.

2.2 Approaches to Characterize System Security

Formal, non-formal, and semi-formal graph-based methods have all been applied during development to improve system security. Formal methods can be applied to specify desired security properties in models [RS+02] and have been used in design phase security analysis to identify vulnerabilities through model checking counterexamples [RA00, ACC+10, BNR03]. Formal methods have been applied to otherwise non-formal techniques such as attack trees, as discussed in a survey by Widel et al. [WAFP19], in order to aid in their analysis and ensure correctness.

Non-formal methods for design-phase security analysis range widely. These include attempts to identify problematic areas of design using a DREAD-based threat modelling approach [MAM+20], and Monte-Carlo sampling to randomly simulate attacks in a model [WCDM+18]. Cukier and Panjwani [CP09] attempt to predict and prioritize vulnerabilities by using real-world data collected with honeypots, while Hadar and Hassanzadeh [HH19] compare previously identified threats to attack graphs of a new system to generate and prioritize remediation requirements. Bakirtzis et al. [BSC+19]

draw on existing databases of exploits to locate possible attack vectors in a system model. Though all these methods benefit from leveraging real world data, this data is not necessarily an exhaustive list of vulnerabilities and only considers what has been exploited so far. Moreover, to collect meaningful data, a honeypot needs to be similar to the system to be built, which may require a significant amount of development.

Implementation-phase metrics for coupling and complexity have been demonstrated on system implementations as methods to predict vulnerabilities and improve security [SW08, MSA13]. Gegick et al. [GWOV08] also use code-based metrics to prioritize components for redesign, but all these methods require the system implementation to be complete. Analysis of code rather than a system model can be beneficial, as there is no concern with model validity. However, such an approach requires the system to have been built, whereas analysis of a model means the analysis can be performed earlier in the design phase of the project.

Formal methods can provide high assurance and unambiguous results [CGD+16] but they suffer from scalability issues and doubts over their claimed cost-effectiveness [Hei98], and often require significant expertise to use [Man03]. If a system model exists, measurements from that model can be much more easily collected and automated, but do not have the same rigor as formal methods. Both methods, being rooted in mathematics, offer objectivity and repeatability to security analyses. Currently only formal methods exist to identify implicit interactions, but while we plan to extend those formal methods as per operator feedback in [Jas20b], we also seek to lower the level of expertise required to consider this class of vulnerability at the design phase by

looking for system measurements that provide results similar to what the existing formal methods yield.

2.3 Graph-Based Methods in System Development

Graph-based methods have been applied to aid in adding formality to the evolution of a system design [LM96, HIM98] and to track dependencies between software modules [PC90]. UML diagrams, such as class diagrams, can be used in a graph-based setting to help validate consistency between design and application [BHTV03] and to identify constructs such as design patterns or groupings of classes [CTS06].

Younis et al. [YMR16] employ a reachability analysis on source code to find "dangerous system calls" and assign each a vulnerability score with the Common Vulnerability Scoring System. Like the non-formal implementation-phase methods discussed above, this requires complete source code, meaning changes are more expensive to make to the system to resolve these vulnerabilities [DBRB10].

In the modelling of systems, graph-based approaches have been used to measure a number of system properties and locate areas for improvement. Mortula et al. [MAS+20] discuss the relationship between degree centrality and resilience for identifying vulnerabilities, Balakrishnan et al. [BZ20] explore node criticality and node susceptibility in a physical infrastructure network to locate resilience issues, and work by Samuel et al. [SJY21a] includes the use of eigenvector centrality to help characterize the structural security posture of a system. Graph-based methods have also been applied to modelling other phenomena such as cascading failures [YCM20], and have

been used to develop metrics to characterize the overall resilience of cyber-physical systems [BCC+20] and systems of systems [HMD12]. For object-oriented development, Alshammari et al. [AFC09] suggest measurements based on the number accessors and mutators for a given system environment to capture a component's attack surface, and Aggarwal et al. [ASKM07] present component-level measurements based on the number of exceptions required for each component.

At least one previous work exists that uses graph-based analyses to approximate system properties in formal specifications. Miller and Chaudry approximate the performance of formal protocol models using graph-based techniques [MC99]. What remains to be seen is whether we can leverage similar graph-based approaches to help reduce the prevalence of implicit interactions in a system design, and whether we can approximate the results obtained from the formal C^2KA specification of a system. Specifically, graph-based measures such as degree centrality used in [MAS+20] and eigenvector centrality used in [SJY21a] may be also usable for locating areas of a design to modify to reduce the prevalence of implicit interactions, and measurements discussed in other works may relate to how the prevalence of implicit interactions changes from one system design to another. Showing such a relationship exists can help further solidify these as strong design-level measurements for less formal security analyses.

2.4 Design Principles for Improving System Quality

Design principles embody best practices of system development and serve as the foundation for a strong design and implementation [SS75]. Principles are developed through research or experience, and are intended to have wide applicability. Specific design principles exist for targeting specific quality attributes, such as performance [DAD+04], reliability [RG12], and security [SS75, SB18], but also for the general maintainability of systems to avoid creeping complexity and improve flexibility in systems prone to change [Mar00]. The level of applicability can also widely vary, from design principles for software systems in general [QFTX10], principles for specific paradigms of software development [Mar00, MNK03], or principles meant to be applicable to systems in general whether composed of software, physical components, or a combination [RG12].

Previous works exist to show the applicability of specific security design principles to a problem domain through case study systems [MM20, VGM+20] and informally evaluate the applicability of security design principles to processes such as CLASP [BSJ07]. Other works works have also formally evaluated the applicability of design principles for preventing specific types of attacks (e.g., Sybil attacks in social networks [Fon11], replay attacks in protocols [LPMH07]). Alshammari et al. [AFC09] explicitly use security principles to inform which graph-based measurements they explore in system models.

Previous work on implicit interactions has posited that implicit interactions come from the hidden complexity and coupling in system designs [JV17a, Jas20a], so we would like to understand how more general design principles related to ideas such as coupling and cohesion [QFTX10] play a role in the prevalence of implicit interactions in a system design. We are particularly interested in the list of fundamental security design principles described in [SB18], many of which originally appeared in work by Saltzer and Schroeder [SS75] and have endured over the years despite numerous changes in the security landscape [Smi12]. While this long-lasting list of design principles are thought to be widely applicably, it is currently unclear which principles relate to implicit interactions and in what way. We can apply a method similar to Alshammari et al. [AFC09] to link measurements and design principles, but in reverse. Having evaluated what graph-based measurements relate to the formal method results from previous works, we can then explore where the concerns of design principles overlap with the design concerns that various graph-based measurements attempt to quantify.

2.5 Conclusions

In this chapter we explored various approaches to improving systems designs with varying levels of formality, including current work up to this point on identifying and analyzing implicit interactions. The main takeaways from this chapter are:

1. Formal methods exist for characterizing security, and specifically implicit interactions, but formal methods as a whole see limited adoption.

2. Measurements from models, including graph-based measurements, are used in many settings and enable security analysis with less formal expertise. None so far have been proposed to capture the notion of implicit interactions.

3. Design principles and quality attributes are much more abstract concepts that do have established relationships with security, but it is unclear how they relate specifically to implicit interactions.

To the best of our knowledge, no works exist that combine all three of formal methods, graph-based methods, and design principles to improve the security of system designs. In subsequent chapters, we detail our approach which encompasses all three methods to provide many approaches for considering the prevalence of implicit interactions at the design phase of system development.

Chapter 3

Overview of the Approach

In this chapter, we present a research approach rooted in formal modelling and mathematical analysis in order to answer the research questions posed in Section 1.3 by discovering design principles, system properties, and measurements that are relevant to the prevalence of implicit interactions in a system design. The principles, properties, and measurements discovered are intended to be directly applicable to a system design development phase in order to help reduce the prevalence of implicit interactions in a system design without requiring expertise in the formal methods that are currently needed to identify implicit interactions. We break the research activities into three main sections: data collection activities, shown in Figure 3.1, and activities related to the analysis of said data and the interpretation of those results, shown in Figure 3.2. Figure 3.1 shows how different aspects of a system design (left) are used in research activities (center) to generate data (right) for the rest of the activities. Figure 3.2 shows how the same data (left) is analyzed and interpreted

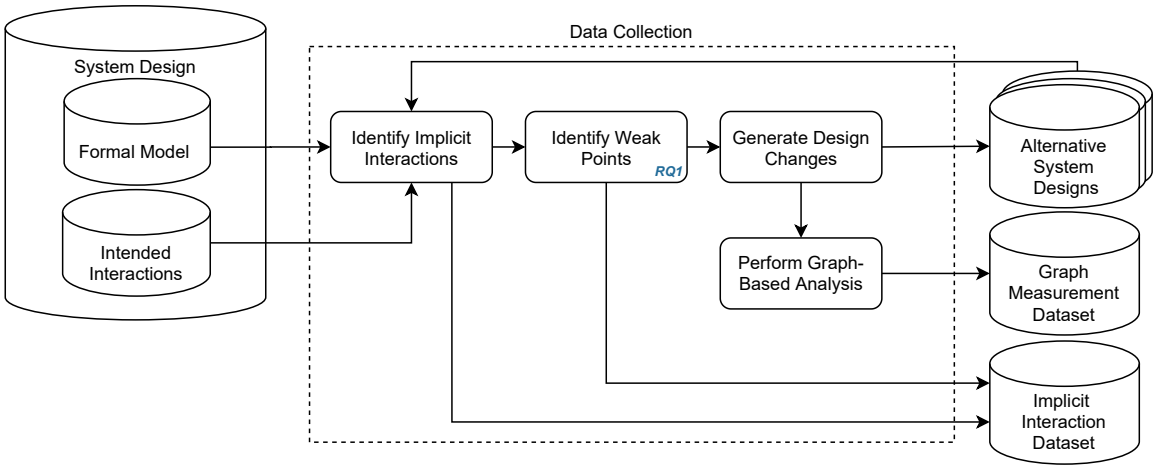


Figure 3.1: High-level view of the research activities conducted to generate data for analysis activities

(center), and how the results of the research activities can be used in a system design development phase (right). In both figures, solid arrows denote how research activities (rounded rectangles) relate to each other and to the inputs and outputs of the overall research approach (cylinders). The dashed line between the research outputs **Relevant System Properties** and **Relevant Quality Attributes** denotes that the quality attributes are derived from those relevant system properties.

3.1 System Models Required

For the activities we perform, we need a system design that contains a **formal model**, and the **intended interactions** that describe the system's expected behaviors. For the formal model, we use C^2KA in order to adopt the method and tools for identifying implicit interactions from previous work [Jas15, JV17b]. The intended interactions

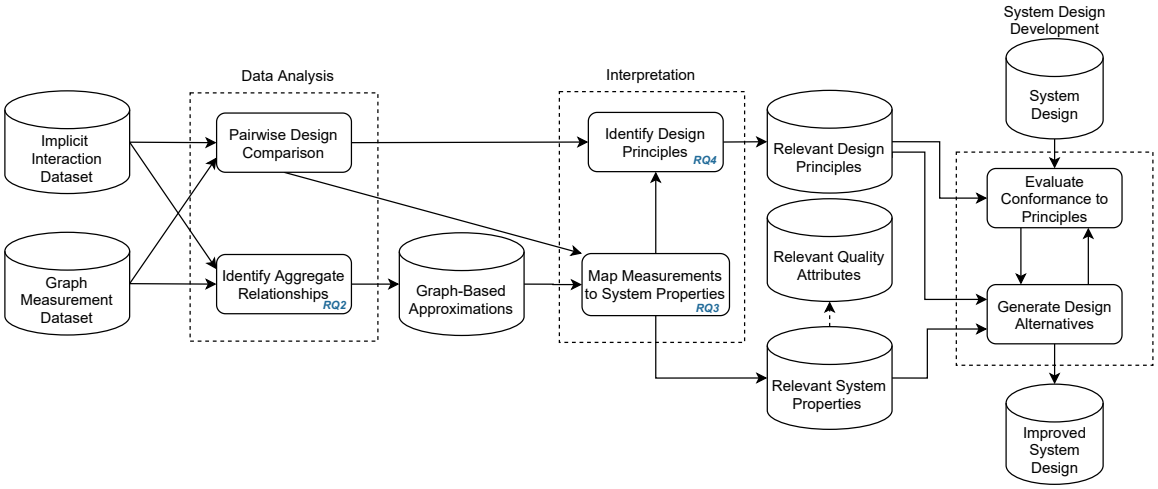


Figure 3.2: High-level view of the research activities conducted to analyze data and interpret results, and how research output relates to system development activities

of a system are the communication pathways that represent desired system functionality [JV17b] and are determined by the system description and requirements.

In this work, we use multiple case study system designs that were developed and verified in previous works [JV17b, JV17a, Jas20b]. Our aim is to make design changes to these models to answer our research questions posed in Section 1.3. The case study system designs and the different system views required are detailed in Chapter 4.

3.2 Data Collection Activities

The C^{2}KA model and intended interactions of a system design are used to **identify implicit interactions** that exist. This is done through the method and tool support provided in [JV17b]. This produces the set of implicit interactions $\mathcal{P}_{implicit}$ for a given system.

Next, we use the implicit interactions, along with the formal model from which they are generated, to determine what part(s) of a system design should be modified to try to reduce the prevalence of implicit interactions in that system. Any design change can introduce new implicit interactions as well as remove existing ones, so we want to determine which agents or communication channels between agents contribute highly to the current implicit interactions in the system design in order to ensure we are not just make the situation worse. We use the term *weak points* to refer to these parts of the system design that contribute highly to the prevalence of implicit interactions. To **identify weak points**, we define a function to produce a value that tells us how "weak" a given agent or communication channel of a system is in the context of implicit interactions, with weaker parts having higher prioritization for design change. The implicit interactions and weak points identified in each system design for which we perform these formal analysis activities are compiled into an **implicit interaction dataset** for use in later analysis activities. The method to identify weak points answers research question **RQ1**. in Section 1.3 and constitutes our research contribution **C1**. presented in Section 1.4. The method to identify weak points is discussed in detail in Section 5.2.

Given the identified weak points, we then **generate design changes**. As we do this work at the design phase, and implicit interactions deal with the *potential* for communication, we are primarily interested in structural changes to the system and changes to what is communicated, rather than behavioral changes to implement something more akin to traditional countermeasures such as a firewall or filter for messages. Of course, changes to the system structure and communication can have knock-on effects for the system behavior, but this is not the focus of those design changes. When this

occurs, we informally show that the system is still able to perform its intended interactions, and whether any changes to these communication sequences need to be made to satisfy the system's objectives. As we are interested in the system properties that are related to the prevalence of implicit interactions in a system, we also generate design changes based around manipulating specific system properties. This activity produces new **alternative system designs**, allowing application of the overall methodology in a iterative fashion.

For each alternative system design generated, we **perform graph-based analysis** on the system design by generating graph-based representations from which to take measurements related to the structure and ordering of communications in the design. The graph-based measurements taken have been related to different system properties in other works, allowing us to track these system properties across the different design alternatives. The measurements taken are compiled into a **graph measurement dataset** for use in analysis activities. The graph-based representations used and a description of the measurements taken is given in Section 5.4.2.

3.3 Data Analysis Activities

The results of the formal analyses in the **implicit interaction dataset** and the graph-based analysis in the **graph measurement dataset** are used in two ways.

First, we **identify aggregate relationships** between the graph-based measurements and the existing implicit interactions and identified weak points. We are looking for correlations between the formal analysis results and graph-based analysis to see if

we can find measurements that reliably increase or decrease as the prevalence of implicit interactions in the system changes. Graph-based measures that show strong relationships can be used to approximate the formal analysis, which allows for implicit interactions as a design concern to be considered by a wider range of practitioners by avoiding issues related to cost, scalability, and required expertise that often plague formal methods [Hei98, Man03]. We consider both relationships observed across every system design, and relationships observed within each specific design across the components in the system. The **graph-based approximations** produced by this activity provide an answer to research question **RQ2.** in Section 1.3 and provides the "measurement" portion of contribution **C2.** in Section 1.4.

Second, we do a **pairwise design comparison** of the data for alternative system designs. This allows us to identify examples of specific design changes that either increase or decrease the prevalence of implicit interactions within the system, while also impacting specific quality attributes and system properties, or showing improved conformance to a design principle. The pairs of designs identified here help to provide concrete examples of how the application of design principles can help to reduce the prevalence of implicit interactions in a system, and how a design change can impact a systems quality attributes.

3.4 Interpretation Activities

We **map measurements to system properties** by reviewing existing literature on the quantification of system properties and quality attributes. This produces a set

of system properties that relate strongly to the prevalence of implicit interactions in the system, and includes the quality attributes that we can expect to be impacted if design changes are made in a system if we want to make changes to other systems to reduce the prevalence of implicit interactions in its design. The system properties identified in this activity answer research question **RQ3.** in Section 1.3 and provide the system property and system quality attribute portions of contribution **C3.** in Section 1.4.

We use these system properties and design comparisons to **identify design principles** that are in line with our research goals of mitigating the prevalence of implicit interactions in a system design. We use concrete examples from the generation of design alternatives as evidence to show how application of identified principles impacts the prevalence of implicit interactions and map identified system properties to additional design principles where possible. The design principles identified here answer research question **RQ4.** in Section 1.3 and provide the design principle portion of contribution **C3.** in Section 1.4.

3.5 Practical Applicability of Research Approach

The right-side of Figure 3.2 shows an example of the activities conducted in the design phase of a system's development and how the results of the research activities can be used in the development process. Given a system design, which does not necessarily require a formal model, we can use the results of the research approach to **evaluate conformance to principles** that we identify as being related to the prevalence of

implicit interactions. If we find a lack of adherence to those design principles, we can **generate design alternatives** with those relevant design principles in mind with the goal of making changes to reduce the prevalence of implicit interactions without mathematical analysis. Our work in identifying system properties that are relevant to implicit interactions helps designers to be aware of what impacts to the system may occur through their design changes, which is useful when they weigh tradeoffs between different design alternatives. This can be done in an iterative fashion until the designers are satisfied with conformance to design principles, ultimately producing an improved system design with a lower prevalence of implicit interactions.

While this work aims to make mitigating the threat posed by implicit interactions a more widely applicable practice, situations in which formal analysis is required already can also make use of our research approach, as shown in Figure 3.3. From a formal model of a system and the intended interactions, a designer can **identify implicit interactions** in the model and use these to **identify weak points**. The weak points can be used to help guide designers to **generate design changes** as transformations to the formal model. Then, the designer can **compare designs** to be aware on any tradeoffs or knock-on effects the transformation produces, resulting again in an improved and more informed system design.

A formal analysis might be required in situations where the high assurance and unambiguous results they provide [CGD+16] are necessary, such as when a system needs to comply with high evaluation assurance levels as set out by the common criteria standards [Com17]. Still, if a formal analysis is required, iterative changes could also

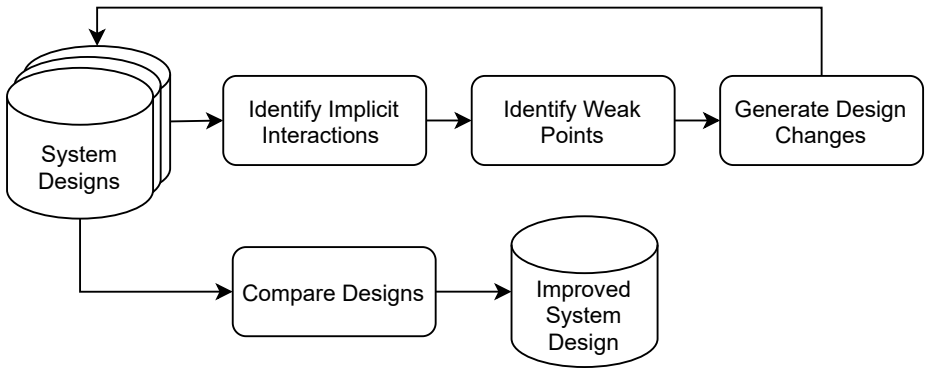


Figure 3.3: High-Level View of Research Activities Usable in Formal Industry Development

be made with the identified graph-based approximations, system properties, and design principles. Rather than repeating a full formal analysis every design iteration, using the semi-formal graph-based methods or non-formal design principles to guide redesign efforts, and then performing formal analysis at the end of the design phase can help with shrinking the time between iterations meeting any requirements for use of formal methods. After several design iterations on a case study system without employing formal methods, we formally identify implicit interactions and weak points to verify whether the application of identified design principles produces similar design changes to what the formal methods suggest.

3.6 Conclusions

This chapter serves to familiarize readers with the research approach and how it produces practical results that can be used in future system development. The main takeaways from this chapter are:

1. We extend the existing formal method-based approaches for analyzing implicit interactions to guide where redesign efforts should be focused to reduce the prevalence of implicit interactions in the systems under study.

2. We use this method to produce several design alternatives of systems, from which we collect data related to the new and existing formal method-based approaches and several graph-based approaches to identify graph-based approximations of the formal methods and abstract results away from those formal methods.

3. We further abstract away from formal methods by using those identified graph-based approximations with comparisons of design alternatives to identify system properties and design principles that are relevant to the prevalence of implicit interactions in a system design.

Next, after introducing the system designs used as illustrative examples in this work, subsequent chapters detail each activity in the approach described in this chapter to answer our research questions.

Chapter 4

System Designs

In this chapter, we describe in detail the system designs used as illustrative examples to answer our research questions, the formalism used, and relevant terminology and notations. In Section 4.1, we describe the formalism used to model system designs. In Section 4.2 and Section 4.3, we introduce terminology used throughout this work. In Section 4.4 and Section 4.5, we describe our two illustrative examples: a manufacturing cell control system and a wastewater dechlorination system.

4.1 Communicating Concurrent Kleene Algebra

We use the algebraic modelling framework known as Communicating Concurrent Kleene Algebra (C^{2}KA) [Jas15, JKZ14] for system modelling. C^{2}KA relates the Concurrent Kleene Algebra (CKA) [HMSW09] $\mathcal{K} = \big(K, +, *, ;, \circledast, \odot, 0, 1\big)$, which contains a set of behaviors K, and the idempotent semiring $\mathcal{S} = \big(S, \oplus, \odot, \mathfrak{d}, \mathfrak{n}\big)$, which

contains a set of stimuli S, through a next-behavior mapping $\circ : K \times S \to K$ and next-stimulus mapping $\lambda : K \times S \to S$. This modelling framework has been used in several published works related to implicit interactions [JV17b, JV17a, Jas20b, NJ21], and has existing tool support [Jas15, JV17b] that can be used for our purposes. This system offers three levels of system specification: abstract behavior specification, stimulus-response specification, and concrete behavior specification. In this section, we focus on describing these three levels of specification as is relevant for this work; a complete description of C^{2}KA is detailed in [Jas15].

The abstract behavior specification represents each agent in a system as an algebraic term restricting the agent to specific behaviors in the system (e.g., Figure 4.2). This utilizes the CKA $\mathcal{K} = \big(K, +, *, ;, \circledast, \circledcirc, 0, 1\big)$ for a particular system, where K is the set of all possible behaviors in the system, $+$ is a choice between two behaviors, $*$ and $;$ are parallel and sequential composition of behaviors, and $\circledast$ and $\circledcirc$ are parallel and sequential iteration of behaviors. 0 and 1 are special elements of K that represent the behavior of an inactive agent and idle agent.

The stimulus-response specification focuses on specifying the next-behavior mapping $\circ : K \times S \to K$ and next-stimulus mapping $\lambda : K \times S \to S$. The next-behavior mapping describes how agents' behaviors change as a result of their current behavior and incoming stimuli, and the next-stimulus mapping describes whether an agent exhibiting a specific behavior will broadcast stimuli in response to receiving other stimuli. These are shown as tables (e.g., Table 4.1) where columns correspond to each stimuli, rows correspond to each behavior, and each cell is the new behavior of the agent or stimulus broadcast by the agent in response to receiving the column

stimulus in the row behavior. The neutral stimulus $\mathfrak{n}$ is used in the next-stimulus mapping when the agent would not send a message in response to receiving a stimulus when exhibiting a specific behavior.

The concrete behavior specification further specifies agents' behaviors using Dijkstra's guarded command language [Dij75], which allows for a state-level specification of how local and shared environments are defined and referenced (e.g., Figure 4.3). Each behavior in the set K can be expanded upon in the concrete behavior specification to describe the actions taken by an agent when it exhibits each behavior that constitutes it's algebraic term in the abstract behavior specification.

4.2 Agents and Agent Communication

The set of all agents in a system is denoted by $\mathcal{A}$. We associate a specific agent (e.g., $\mathsf{A} \in \mathcal{A}$) with it's abstract behavior specification (e.g., $a \in K$) with the notation $\mathsf{A} \mapsto \langle a \rangle$ to denote A's actions are restricted to only the behavior a. Each interaction of an agent with its neighbouring agents is called a *communication* [Mil89]. Interactions between agents are categorized into either communication *via stimuli* (e.g., sending or receiving messages), denoted $\rightarrow_{\mathcal{S}}$, or communication *via shared environments* (e.g., reading or writing shared variables), denoted $\rightarrow_{\mathcal{E}}$. Communication via stimuli is the focus of the stimulus-response specification, while communication via shared environment is specified in the concrete behavior specification. Generally,

agent interactions of any length or type are written as:

$$\mathsf{A}_n \rightarrow_{\mathcal{T}_n} \mathsf{A}_{n-1} \rightarrow_{\mathcal{T}_{n-1}} \ldots \rightarrow_{\mathcal{T}_2} \mathsf{A}_1 \rightarrow_{\mathcal{T}_1} \mathsf{A}_0 \tag{4.1}$$

where $\mathsf{A}_i \in \mathcal{A}$ for $0 \leq i \leq n$, A_n is the *source agent*, A_0 is the *sink agent*, and $\mathcal{T}_i \in \{\mathcal{S}, \mathcal{E}\}$ for $1 \leq i \leq n$ [JV17a]. Each $\mathsf{A}_i \rightarrow_{\mathcal{T}_i} \mathsf{A}_{i-1}$ represents a *direct* interaction between two agents. The set of all direct interactions is denoted as $\mathcal{P}_{direct}$. An *indirect* interaction (denoted $\mathsf{A} \rightarrow^+ \mathsf{B}$ for $\mathsf{A}, \mathsf{B} \in \mathcal{A}$) on the other hand, is communication pathway that passes through one or more intermediate agents between the source and sink. In general, an interaction can be described as a walk of the communication graph of a system design [JV17a], as they alternate between the agents and interactions of a system design and can repeat an agent any number of times.

4.3 Intended Interactions

The intended interactions of a system, denoted as $\mathcal{P}_{intended}$, is the set of all communication pathways that represent desired system functionality [JV17b], as determined by the system description and requirements. These may come from behavioral diagrams of a system design such as a collaboration diagram or sequence diagram, or may be extracted from a text description of the actions conducted to achieve a system's objectives. Rather than list all the interactions that compose $\mathcal{P}_{intended}$ in the form presented in Equation 4.1, we generally present the intended interactions as an event trace diagram (e.g., Figure 4.4), where solid arrows represent the communication via stimuli, and dashed arrows represent communication via shared environments.

Multiple intended interactions can exist in a system design due to concurrent actions performed, and through previously defined shared environments later referenced in the control flow of a system. As the intended interactions are the expected behaviors of the system and are the focus of the system design, other communication pathways may not be considered to the same degree during development. These other paths, the implicit interactions, are all the potential communication paths in the system that are not sub-paths of any walk in $\mathcal{P}_{intended}$ [JV17a].

4.4 Illustrative Example: Manufacturing Cell Control System (MCCS)

This model of a distributed Manufacturing Cell Control System (MCCS) is adapted from [GZY+01]. This version of the model is developed originally in [JV17b], and has been used in other works related to implicit interactions [JV17a, NJ21].

4.4.1 System Description

The system is composed of four agents: *Control Agent* C, *Storage Agent* S, *Handling Agent* H, and *Processing Agent* P. The *Control Agent* C is responsible for coordinating the activities of the other agents in the system, and for maintaining the overall system state. The *Storage Agent* S is responsible for storing materials required for the manufacturing assembly, and for maintaining a record of its empty/full status. The *Handling Agent* H is responsible for moving the materials from storage so that

they can be processed, and for recording the readiness of the material for processing. Lastly, the *Processing Agent* P is responsible for processing the material to its manufactured state.

The operation of the MCCS can be visualized as shown in the collaboration diagram given in Figure 4.1, where the solid arrows denote message-passing communication and the dashed arrows denote shared variable communication. Note that Figure 4.1 depicts only the communication between agents and the abstract behaviors of each agent. It does not show the detailed behavior of each agent. The operation of the system is initiated from outside the system boundary, which is represented by the dashed square around the agents. The system boundary defines the scope of the system. Communication that crosses this boundary may come from other systems, such as other industrial control systems or the organization's IT systems, or can come from operators interacting with the system, such as the press of a button.

The manufacturing process begins with a *start* event (1) from outside the system boundary. C begins the manufacturing process by broadcasting a *load* event and assigning the shared variable `state` the value 1 to indicate that the system is loading the material. S responds to the *load* event (2) by loading the material and assigning the shared variable `status` the value 1 to indicate the storage is full. When the loading is complete, S broadcasts a *loaded* message (3).

C responds by assigning the shared variable `state` the value 2 to indicate the system is preparing the material for processing. It also transmits a *prepare* event (4). H responds by verifying that the material storage is loaded (i.e., by checking that `status` = 1 (5)) before assigning the shared variable `material` the value 1 to indicate that the material

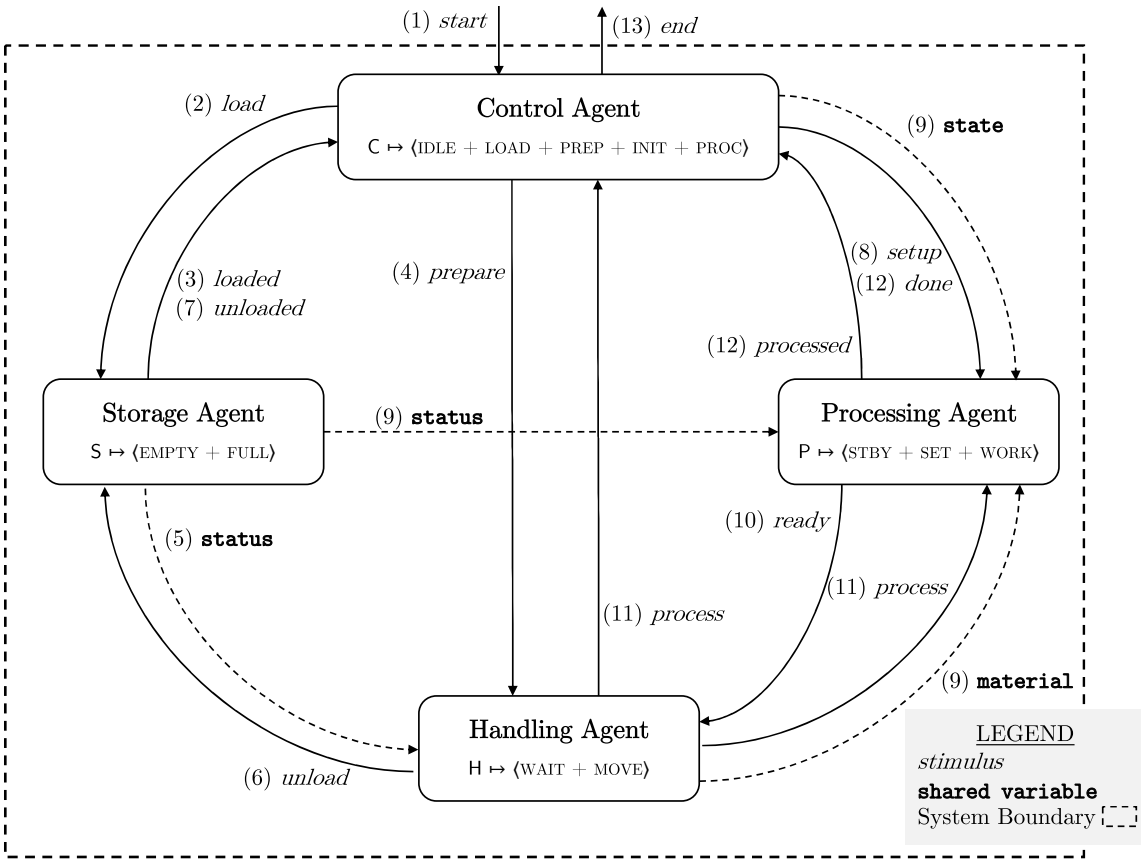


Figure 4.1: Collaboration diagram depicting the expected behavior of the MCCS

is ready to be moved for processing. If the material storage is not loaded, then `material` is assigned the value 0. H also sends an *unload* event (6) to which S responds by entering its unloading the material and assigning `status` the value 0 to indicate that the storage is now empty.

After unloading, S broadcasts an *unloaded* message (7) that causes C to respond by assigning `state` the value 3 to indicate the system is initializing, and a *setup* event (8) is issued. P responds by ensuring that the material is ready to be processed (9), the system is initialized (9), and the material storage is empty (9). If the condition is satisfied, then the shared variable `ready` is assigned the value 1 to indicate that the

system is set for processing; otherwise, `ready` is assigned the value 0. P then transmits a *ready* message (10) which causes H to transition to its waiting behavior and to broadcast a *process* event (11). Both P and C respond to the *process* event. P verifies that the material is ready for processing (i.e., by checking that `material` = 1) before executing the PROCESS() procedure. If the material is not ready for processing, then P does not process the part (i.e., `part` is null). Concurrently, C assigns `state` the value 4 to indicate the system is processing the material. When C is finished, it issues a *done* event (12) that causes P to return to its standby behavior. Similarly, once P has finished working, it issues a *processed* event (12) that causes C to assign `state` the value 0 to indicate the system is idle. C then broadcasts an *end* message (13) that may indicate to other connected systems that the manufacturing process is complete. The system then awaits another *start* event to begin the manufacturing process again.

4.4.2 Abstract Behavior Specification

Figure 4.2 shows the abstract behavior specification of each agent in the MCCS. For example, the abstract behavior for the agent S in the MCCS is $\mathsf{S} \mapsto \langle \textsc{empty} + \textsc{full} \rangle$ to restrict the operator to either the behavior EMPTY, which denotes the storage agent does not have any material loaded, or the behavior FULL, which denotes that the storage agent currently has material loaded.

Due to the simplicity of the MCCS, we only require the + operator to describe the abstract behaviors. More complex behaviors in other systems can necessitate the use of the other operators in the CKA.

$$
\begin{aligned}
\mathsf{C} &\mapsto \langle \textsc{idle} + \textsc{load} + \textsc{prep} + \textsc{init} + \textsc{proc} \rangle \\
\mathsf{S} &\mapsto \langle \textsc{empty} + \textsc{full} \rangle \\
\mathsf{H} &\mapsto \langle \textsc{wait} + \textsc{move} \rangle \\
\mathsf{P} &\mapsto \langle \textsc{stby} + \textsc{set} + \textsc{work} \rangle
\end{aligned}
$$

Figure 4.2: Abstract behavior specification of the MCCS agents

4.4.3 Stimulus Response Specification

Table 4.1 shows the stimulus-response specification for the next-behavior and next-stimulus mappings for agent S in the MCCS. We can see from Figure 4.1 that S has two behaviors, EMPTY and FULL, which make up the rows of Table 4.1, and the system includes the following stimuli that make up the columns of the stimulus response tables and match to each event described in Section 4.4: {*start*, *load*, *loaded*, *prepare*, *done*, *unload*, *unloaded*, *setup*, *ready*, *process*, *processed*, *end*, $\mathfrak{d}$, $\mathfrak{n}$}. These tables tell us that, for example, when S receives a *load* stimulus and it is in the EMPTY behavior, it changes to the FULL behavior according to the next-behavior mapping of S (the top half of Table 4.1) and broadcasts a *loaded* stimulus according to the next-stimulus mapping of S (the bottom half of Table 4.1).

Table 4.1: Stimulus-response specification of the Control Agent S.

$\circ$	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
EMPTY	EMPTY	FULL	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY	EMPTY
FULL	FULL	FULL	FULL	FULL	FULL	EMPTY	FULL	FULL	FULL	FULL	FULL	FULL

λ	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
EMPTY	$\mathfrak{n}$	*loaded*	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$
FULL	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	*unloaded*	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$	$\mathfrak{n}$

While we can include all behaviors in a single table each for the next-behavior and next-stimulus mappings, we break these down by the agent that exhibits each behavior

to improve readability. The stimulus-response specification for the remaining three agents are given in Table 4.2, Table 4.3, and Table 4.4:

Table 4.2: Stimulus-response specification of the Control Agent C.

$\circ$	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
IDLE	LOAD	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE
LOAD	LOAD	LOAD	PREP	LOAD	LOAD	LOAD	LOAD	LOAD	LOAD	LOAD	LOAD	LOAD
PREP	PREP	PREP	PREP	PREP	PREP	PREP	INIT	PREP	PREP	PREP	PREP	PREP
INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT	INIT	PROC	INIT	INIT
PROC	PROC	PROC	PROC	PROC	PROC	PROC	PROC	PROC	PROC	PROC	IDLE	PROC

λ	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
IDLE	*load*	n	n	n	n	n	n	n	n	n	n	n
LOAD	n	n	*prepare*	n	n	n	n	n	n	n	n	n
PREP	n	n	n	n	n	n	*setup*	n	n	n	n	n
INIT	n	n	n	n	n	n	n	n	n	*done*	n	n
PROC	n	n	n	n	n	n	n	n	n	n	*end*	n

Table 4.3: Stimulus-response specification of the Handling Agent H.

$\circ$	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
WAIT	WAIT	WAIT	WAIT	MOVE	WAIT	WAIT	WAIT	WAIT	WAIT	WAIT	WAIT	WAIT
MOVE	MOVE	MOVE	MOVE	MOVE	MOVE	MOVE	MOVE	MOVE	WAIT	MOVE	MOVE	MOVE

λ	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
WAIT	n	n	n	*unload*	n	n	n	n	n	n	n	n
MOVE	n	n	n	n	n	n	n	n	*process*	n	n	n

Table 4.4: Stimulus-response specification of the Processing Agent P.

$\circ$	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
STBY	STBY	STBY	STBY	STBY	STBY	STBY	STBY	SET	STBY	STBY	STBY	STBY
SET	SET	SET	SET	SET	SET	SET	SET	SET	SET	WORK	SET	SET
WORK	WORK	WORK	WORK	WORK	STBY	WORK	WORK	WORK	WORK	WORK	WORK	WORK

λ	*start*	*load*	*loaded*	*prepare*	*done*	*unload*	*unloaded*	*setup*	*ready*	*process*	*processed*	*end*
STBY	n	n	n	n	n	n	n	*ready*	n	n	n	n
SET	n	n	n	n	n	n	n	n	n	*processed*	n	n
WORK	n	n	n	n	n	n	n	n	n	n	n	n

4.4.4 Concrete Behavior Specification

Figure 4.3 shows the concrete behaviors for the agents in the MCCS. Focusing on the Storage Agent S again, this shows when S enters the EMPTY behavior it sets the

shared environment variable `status` to 0, and when S enters the FULL behavior it sets the shared environment variable `status` to 1. The presence of a skip, such as what is used for when P enters the WAIT behavior, indicates that no actions are taken.

$$
\mathsf{C} \mapsto \begin{cases}
\text{IDLE} \stackrel{\text{def}}{=} \texttt{state} := 0 \\
\text{LOAD} \stackrel{\text{def}}{=} \texttt{state} := 1 \\
\text{PREP} \stackrel{\text{def}}{=} \texttt{state} := 2 \\
\text{INIT} \stackrel{\text{def}}{=} \texttt{state} := 3 \\
\text{PROC} \stackrel{\text{def}}{=} \texttt{state} := 4
\end{cases}
$$

$$
\mathsf{S} \mapsto \begin{cases}
\text{EMPTY} \stackrel{\text{def}}{=} \texttt{status} := 0 \\
\text{FULL} \stackrel{\text{def}}{=} \texttt{status} := 1
\end{cases}
$$

$$
\mathsf{H} \mapsto \begin{cases}
\text{WAIT} \stackrel{\text{def}}{=} \mathsf{skip} \\
\text{MOVE} \stackrel{\text{def}}{=} \mathsf{if} \quad (\texttt{status} = 1) \longrightarrow \texttt{material} := 1 \\
\qquad\qquad [] \quad \neg(\texttt{status} = 1) \longrightarrow \texttt{material} := 0 \\
\qquad\qquad \mathsf{fi}
\end{cases}
$$

$$
\mathsf{P} \mapsto \begin{cases}
\text{STBY} \stackrel{\text{def}}{=} \mathsf{skip} \\
\text{SET} \stackrel{\text{def}}{=} \mathsf{if} \quad (\texttt{material} = 1 \wedge \texttt{state} = 2 \wedge \texttt{status} = 0) \longrightarrow \\
\qquad\qquad \texttt{ready} := 1 \\
\qquad\qquad [] \quad \neg(\texttt{material} = 1 \wedge \texttt{state} = 2 \wedge \texttt{status} = 0) \longrightarrow \\
\qquad\qquad \texttt{ready} := 0 \\
\qquad\qquad \mathsf{fi} \\
\text{WORK} \stackrel{\text{def}}{=} \mathsf{if} \quad (\texttt{ready} = 1) \longrightarrow \texttt{part} := \mathsf{PROCESS}() \\
\qquad\qquad [] \quad \neg(\texttt{ready} = 1) \longrightarrow \texttt{part} := \mathsf{null} \\
\qquad\qquad \mathsf{fi}
\end{cases}
$$

Figure 4.3: Concrete behavior specification of the MCCS agents.

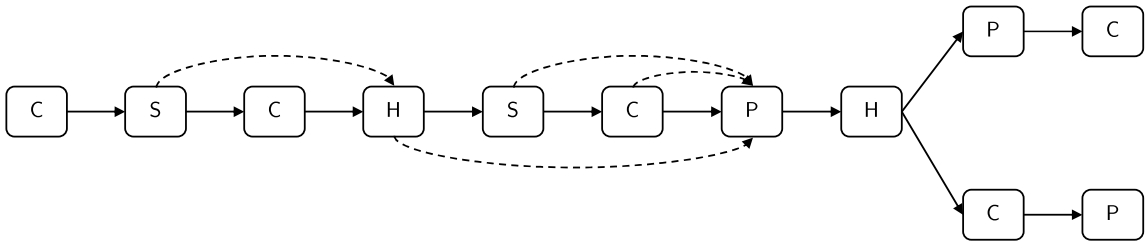


Figure 4.4: Event trace depicting the intended interactions of the distributed MCCS

The concrete behavior specification in Figure 4.3, along with the abstract behavior in Figure 4.2 and stimulus-response specification in Tables 4.1 - 4.4, provides a full C^2KA specification of the MCCS.

4.4.5 Intended Interactions

From the description in Section 4.4 and following the numbering in Figure 4.1, the event trace diagram in Figure 4.4 is constructed.

Different combinations of the concurrency present in the system, such as H broadcasting *process* which triggers behavior changes in both C and P, and referencing of previously defined shared environments, such as `material` which H defines and P references later in the control sequence, result a total of 16 different intended interactions that make up the control flow of the MCCS. For example, one such intended interaction in the system is $\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$, which contains all stimuli triggered in order for the whole control flow. Another is $\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$, which is a shorter overall path as it "jumps" over several of the agents in Figure 4.4 by traversing the communication via shared environment in the system design.

4.5 Illustrative Example: Wastewater Dechlorination System (WDS)

This system design represents a real-world Wastewater Dechlorination System (WDS), and is introduced and discussed in great detail in [Jas20b]. The model in its entirety is developed and verified in [Jas20b] with the aid of operators employed at the facility which hosts the system.

4.5.1 System Description

The system is composed of six agents, which work together as a control system to sample water, perform some analysis, and adjust a pumping rate to maintain a control variable within an acceptable range. The process begins with a *start* event from outside the system boundary. A Sample Pump (SAP) feeds the effluent to an SO_3 Analyzer (SO3) and Sample Flow Meter (SFM) through the shared environment `effluent` and sends the stimulus *eff* to trigger analyses to begin in SO3 and SFM concurrently. SO3 measures the residual SBS in the effluent and sends its results to a Programmable Logic Controller (PLC) through the shared environment `residual` and notifies PLC with a *res* stimulus. Simultaneously, SFM measures the flow rate and sends its results as stimulus *rate* and shared environment `flowRate` to PLC. PLC controls two chemical feed pumps in a lead-lag configuration. PLC's proportional-integral-derivative (PID) control algorithm calculates the SBS flow (in L/h), and determines if the SBS flow should be increased or decreased based on the SBS residual feedback received from SO3. PLC always runs the Lead Pump (CFP1) to match the SBS flow,

up to a maximum flow of 75 L/h. When the SBS flow exceeds the Lead Pump's maximum flow, the PLC activates the Lag Pump (CFP2). Both CFP1 and CFP2 are controlled by the PLC, through shared environments `leadFlow` and `lagFlow` respectively, to meet the SBS flow. When the SBS flow falls below 70 L/h, the PLC deactivates CFP2.

Faults can occur in either SO3 or SAP, represented by a *fault* event incoming to either agent from outside the system boundary. When this occurs, the *fault* stimulus is propagated to the PLC, which sends the stimuli *alarm* to a seventh agent, Operator (OP). While the fault is unresolved, PLC switches from PID control mode (PID) to a Ratio control mode (RATIO) and OP switches from a monitoring behavior (MONITOR) to an alarmed behavior (ALARM). The faulting agent, SO3 or SAP, upon the occurrence of a fault, transitions to a fail state. When the fault is resolved, OP notifies the PLC with a *fixed* stimulus. PLC propagates *fixed* to SO3 AND SAP to return them to their operational states.

The structure of the WDS communications is shown in Figure 4.5. Note that unlike Figure 4.1, Figure 4.5 does not contain the order of communications due to the fact that we have two separate control flows in the system design.

4.5.2 Abstract Behavior Specification

Figure 4.6 shows the abstract behavior specification of each agent in the WDS. Like with the MCCS, we only require the + operator to specify the behaviors of each agent

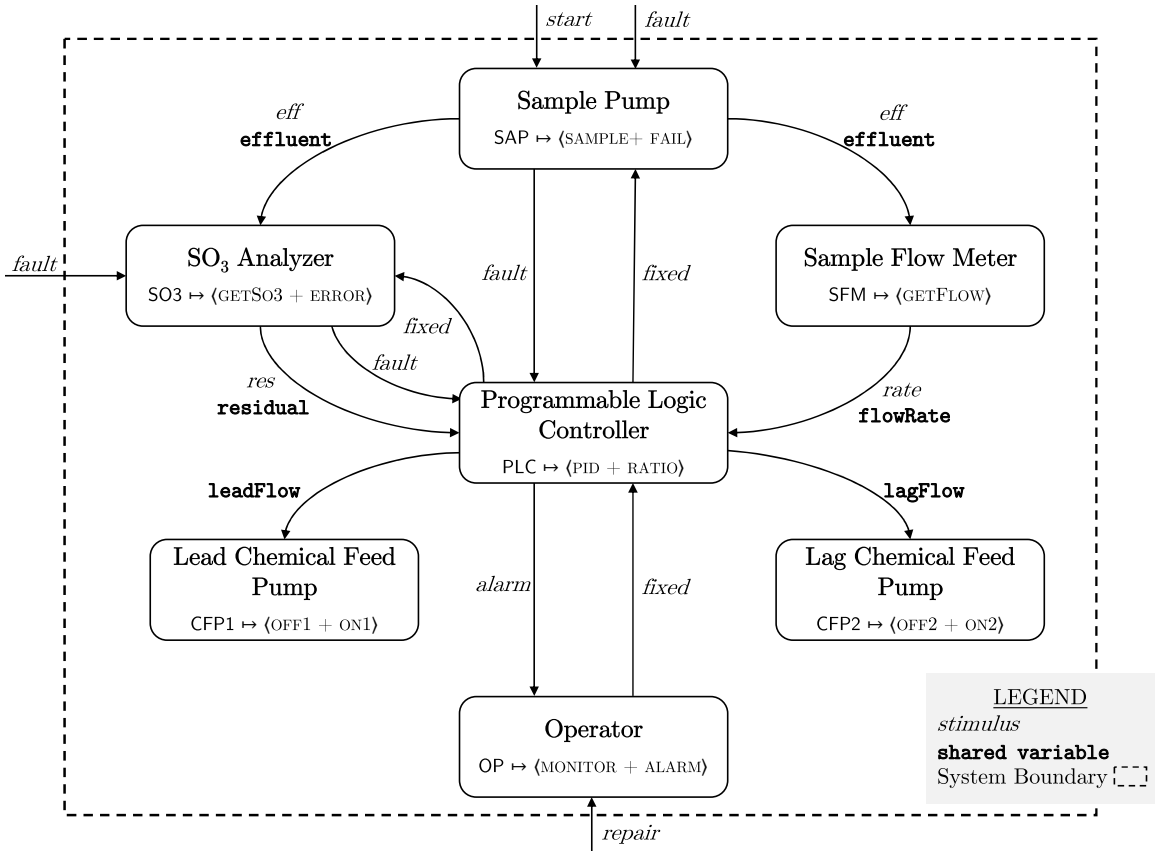


Figure 4.5: Communication graph depicting the agents of the WDS and the stimuli and shared environments they communicate with each other

in this system. The abstract behavior specification of each agent in Figure 4.6 is also shown in Figure 4.5.

4.5.3 Stimulus Response Specification

Tables 4.5 through 4.11 provide the stimulus response specification for each agent in the WDS. The columns in each table correspond to the stimuli shown in Figure 4.5, and the rows correspond to the behaviors shown in the abstract behaviors of each

$$
\begin{aligned}
\mathsf{SAP} &\mapsto \langle \textsc{sample} + \textsc{fail} \rangle \\
\mathsf{SO3} &\mapsto \langle \textsc{getSO3} + \textsc{error} \rangle \\
\mathsf{SFM} &\mapsto \langle \textsc{getFlow} \rangle \\
\mathsf{PLC} &\mapsto \langle \textsc{pid} + \textsc{ratio} \rangle \\
\mathsf{CFP1} &\mapsto \langle \textsc{off1} + \textsc{on1} \rangle \\
\mathsf{CFP2} &\mapsto \langle \textsc{off2} + \textsc{on2} \rangle \\
\mathsf{OP} &\mapsto \langle \textsc{monitor} + \textsc{alarm} \rangle
\end{aligned}
$$

Figure 4.6: Abstract behavior specification of the WDS agents

agent in Figure 4.5 and Figure 4.6. Despite some overlap in naming, the MCCS and WDS are completely independent. For example, even though both illustrative examples have stimuli named *start*, these are two separate stimuli that trigger different actions.

Table 4.5: Stimulus-response specification of Agent SAP

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
SAMPLE	SAMPLE	FAIL	SAMPLE	SAMPLE	SAMPLE	SAMPLE	SAMPLE	SAMPLE	SAMPLE	SAMPLE	SAMPLE	SAMPLE
FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	SAMPLE	FAIL

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
SAMPLE	*eff*	*fault*	n	n	n	n	n	n	n	n	n	n
FAIL	n	n	n	n	n	n	n	n	n	n	n	n

Table 4.6: Stimulus-response specification of Agent SO3

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
GETSO3	GETSO3	ERROR	GETSO3	GETSO3	GETSO3	GETSO3	GETSO3	GETSO3	GETSO3	GETSO3	GETSO3	GETSO3
ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	GETSO3	ERROR

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
GETSO3	n	*fault*	*res*	n	n	n	n	n	n	n	n	n
ERROR	n	n	n	n	n	n	n	n	n	n	n	n

Table 4.7: Stimulus-response specification of Agent SFM

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW	GETFLOW

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
GETFLOW	n	n	*rate*	n	n	n	n	n	n	n	n	n

Table 4.8: Stimulus-response specification of Agent PLC

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
PID	PID	RATIO	PID	PID	PID	PID	PID	PID	PID	PID	PID	PID
RATIO	RATIO	RATIO	RATIO	RATIO	RATIO	RATIO	RATIO	RATIO	RATIO	RATIO	PID	RATIO

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
PID	n	*alarm*	n	n	n	n	n	n	n	n	n	n
RATIO	n	n	n	n	n	n	n	n	n	n	*fixed*	n

Table 4.9: Stimulus-response specification of Agent CFP1

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
OFF1	OFF1	OFF1	OFF1	OFF1	OFF1	OFF1	ON1	OFF1	OFF1	OFF1	OFF1	OFF1
ON1	ON1	ON1	ON1	ON1	ON1	OFF1	ON1	ON1	ON1	ON1	ON1	ON1

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
OFF1	n	n	n	n	n	n	n	n	n	n	n	n
ON1	n	n	n	n	n	n	n	n	n	n	n	n

Table 4.10: Stimulus-response specification of Agent CFP2

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
OFF2	OFF2	OFF2	OFF2	OFF2	OFF2	OFF2	OFF2	OFF2	ON2	OFF2	OFF2	OFF2
ON2	ON2	ON2	ON2	ON2	ON2	ON2	ON2	OFF2	ON2	ON2	ON2	ON2

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
OFF2	n	n	n	n	n	n	n	n	n	n	n	n
ON2	n	n	n	n	n	n	n	n	n	n	n	n

Table 4.11: Stimulus-response specification of Agent OP

$\circ$	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
MONITOR	MONITOR	MONITOR	MONITOR	MONITOR	MONITOR	MONITOR	MONITOR	MONITOR	MONITOR	ALARM	MONITOR	MONITOR
ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	ALARM	MONITOR

λ	*start*	*fault*	*eff*	*res*	*rate*	*off1*	*on1*	*off2*	*on2*	*alarm*	*fixed*	*repair*
MONITOR	n	n	n	n	n	n	n	n	n	n	n	n
ALARM	n	n	n	n	n	n	n	n	n	n	n	*fixed*

4.5.4 Concrete Behavior Specification

Figure 4.7 shows the concrete behaviors for the agents in the WDS. In the concrete behavior specifications of SAP, SO3, SFM, and PLC there are several procedure calls. The names of each procedure describe the actions of an agent when it exhibits a specific behavior, but the details of each procedure are not important. What is important for our level of analysis are the parameters of each procedure as these are

references that result communication via shared environment. For example, the procedures MEASURE_SBS_RESIDUAL and MEASURE_FLOW_RATE both use the parameter `effluent` which is defined by SAP, resulting in the communications via shared environment SAP $\rightarrow_{\mathcal{E}}$ SO3 and SAP $\rightarrow_{\mathcal{E}}$ SFM.

The concrete behavior specification in Figure 4.7, along with the abstract behavior in Figure 4.6 and stimulus-response specification in Tables 4.5 - 4.11, provides a full C^2KA specification of the WDS.

4.5.5 Intended Interactions

From the description in Section 4.5.1, the event trace diagram in Figure 4.8 is constructed. Like the WDS there are multiple intended interactions due to concurrency in the system and communication both via stimuli and via shared environment. The control flow for controlling the chemical feed pumps and managing faults are intertwined in Figure 4.8, which further increases the number of intended interactions to a total of 36 communication pathways.

4.6 Conclusions

The purpose of this chapter was to introduce the terminology and illustrative examples used throughout this work. The main takeaways from this chapter are:

$$
\mathsf{SAP} \mapsto \begin{cases} \textsc{sample} \stackrel{\text{def}}{=} \texttt{effluent} := \mathsf{SAMPLE}() \\ \textsc{fail} \stackrel{\text{def}}{=} \texttt{effluent} := \mathsf{SAMPLE}() \end{cases}
$$

$$
\mathsf{SO3} \mapsto \begin{cases} \textsc{getSO3} \stackrel{\text{def}}{=} \texttt{residual} := \mathsf{MEASURE\_SBS\_RESIDUAL}(\texttt{effluent}) \\ \textsc{error} \stackrel{\text{def}}{=} \texttt{residual} := \mathsf{NULL} \end{cases}
$$

$$
\mathsf{SFM} \mapsto \left\{ \textsc{getFlow} \stackrel{\text{def}}{=} \texttt{flowRate} := \mathsf{MEASURE\_FLOW\_RATE}(\texttt{effluent}) \right.
$$

$$
\mathsf{PLC} \mapsto \begin{cases} \textsc{pid} \stackrel{\text{def}}{=} \begin{array}{l} \mathsf{if} \quad \texttt{flowRate} >= \texttt{FLOW\_SETPOINT} \longrightarrow \\ \qquad \mathsf{skip} \\ [] \quad \texttt{flowRate} < \texttt{FLOW\_SETPOINT} \longrightarrow \\ \qquad \mathsf{send}\ \mathit{alarm} \\ \mathsf{fi}; \\ \texttt{targetFlow} := \mathsf{COMPUTE\_FLOW}(\texttt{residual}); \\ \mathsf{if} \quad \texttt{targetFlow} > \texttt{MAX\_PUMP\_FLOW} \longrightarrow \\ \qquad \texttt{leadFlow} := \texttt{MAX\_PUMP\_FLOW}; \\ \qquad \texttt{lagFlow} := \texttt{targetFlow} - \texttt{MAX\_PUMP\_FLOW} \\ [] \quad \texttt{targetFlow} > \texttt{MAX\_PUMP\_FLOW} \longrightarrow \\ \qquad \texttt{leadFlow} := \texttt{targetFlow} \\ [] \quad \texttt{targetFlow} < \texttt{DEADBAND} \longrightarrow \\ \qquad \texttt{leadFlow} := \texttt{targetFlow} \\ \mathsf{fi} \end{array} \\ \textsc{ratio} \stackrel{\text{def}}{=} \mathsf{skip} \end{cases}
$$

$$
\mathsf{CFP1} \mapsto \begin{cases} \textsc{off1} \stackrel{\text{def}}{=} \texttt{leadPumpOn} := \texttt{FALSE}; \texttt{pumpRate} := 0 \\ \textsc{on1} \stackrel{\text{def}}{=} \texttt{leadPumpOn} := \texttt{TRUE}; \texttt{pumpRate} := \texttt{leadFlow} \end{cases}
$$

$$
\mathsf{CFP2} \mapsto \begin{cases} \textsc{off1} \stackrel{\text{def}}{=} \texttt{lagPumpOn} := \texttt{FALSE}; \texttt{pumpRate} := 0 \\ \textsc{on1} \stackrel{\text{def}}{=} \texttt{lagPumpOn} := \texttt{TRUE}; \texttt{pumpRate} := \texttt{lagFlow} \end{cases}
$$

$$
\mathsf{OP} \mapsto \begin{cases} \textsc{monitor} \stackrel{\text{def}}{=} \texttt{mode} := \texttt{PID} \\ \textsc{alarm} \stackrel{\text{def}}{=} \texttt{mode} := \texttt{RATIO} \end{cases}
$$

Figure 4.7: Concrete behavior specification of the WDS agents.

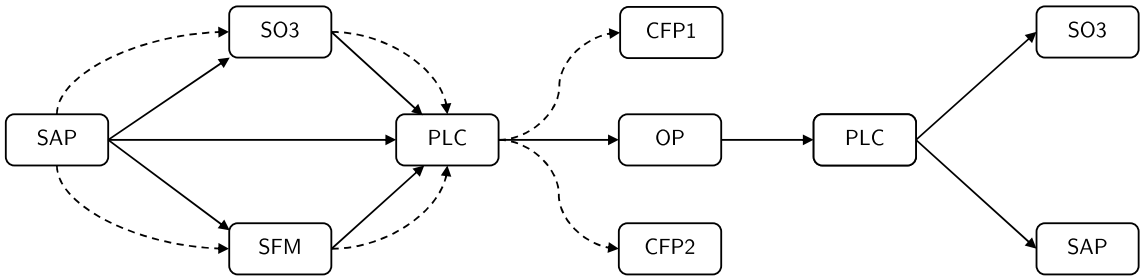


Figure 4.8: Expected execution trace of the intended interactions for the WDS

1. We model systems using C^2KA because this is the modelling framework used for previous work related to implicit interactions [JV17b, JV17a, Jas20a, Jas20b, NJ21], and has some existing tool support [Jas15, JV17b].

2. We provide the formal specifications of two illustrative example systems: a manufacturing cell control system [JV17a] and a wastewater dechlorination system [Jas20b] that will be used throughout the remainder of this thesis.

3. The manufacturing cell control system is the smaller of the two illustrative examples, and models a distributed control flow for processing an abstract "material" into parts.

4. The wastewater dechlorination system is the larger of the two illustrative examples, modelling both the control flow coordinating pumps to maintain a control variable and the control flow for managing faults that may occur in the system. The wastewater dechlorination system model is modelled after a real-world system.

In subsequent chapters, we use the example systems described in Section 4.4 and Section 4.5 to illustrate the activities presented in Chapter 3 to ultimately answer our research questions posed in Section 1.3.

Chapter 5

Data Collection Activities

In this chapter, we detail the activities presented in Chapter 3 to generate data for the analysis activities in Chapter 6. Section 5.1 describes how implicit interactions are identified using existing formal methods [JV17b], Section 5.2 describes how the formal methods are extended to identify weak points in system designs to focus redesign efforts [NJ21] (i.e., answering **RQ1**.), Section 5.3 describes how alternative system designs are generated from the weak points and the system properties we want to test, and Section 5.4 describes the semi-formal graph-based analysis we perform on each alternate system design. In each section, we describe the work conducted in the associated activity, and show their application on our two illustrative examples: the manufacturing cell control system and wastewater dechlorination system.

5.1 Identifying Implicit Interactions

Identifying the implicit interactions in a system design requires the intended interactions $\mathcal{P}_{intended}$, and a C^{2}KA formal model. An implicit interaction is said to exist between agents $\mathsf{A}, \mathsf{B} \in \mathcal{A}$ if and only if [JV17b]:

$$\exists(p \mid p \Longrightarrow (\mathsf{A} \rightarrow^{+} \mathsf{B}) : \forall(q \mid q \in \mathcal{P}_{intended} : \neg SubPath(p, q))) \tag{5.1}$$

where $SubPath(p, q)$ is a predicate indicating that p is a sub-path of q. This means that an implicit interaction p exists if it is a path (i.e., a sequence of nodes without repetition) consisting of direct interactions present in the formal model, but is not a sub-path of any of the walks in $\mathcal{P}_{intended}$.

Using the formal model and intended interactions for the MCCS in Section 4.4 and WDS in Section 4.5, the implicit interactions can be identified with existing tool support [Jas15, JV17b]. The implicit interactions for these models have been identified and presented already for both the MCCS in [JV17a] and for the WDS in [Jas20b], but we need this activity and tool support so that we can identify the implicit interactions in new alternative system designs generated in Section 5.3. In this section, we present all identified implicit interactions to aid with the understanding of this activity, however, in Section 5.3 we only report the prevalence of implicit interactions that is measured as a result of this activity.

We measure the *prevalence of implicit interactions* in a system design by counting how many implicit interactions are identified and dividing by the total number of

potential interactions in a system design to obtain the percentage of potential interactions in a system that are implicit. We also consider just the number of implicit interactions, but this value may be less comparable across system design alternatives. Previous works posit that more complex systems have more implicit interactions [JV17b, JV17a, Jas20a] and if we accept this idea then we expect a system design with more agents or direct interactions between agents to have a larger number of implicit interactions. We test the relationship between complexity and number of implicit interactions in Section 6.2, but using the prevalence of implicit interactions rather than just the number of implicit interactions helps to take into account this complexity and produce a more independent measure. Implicit interactions as a percentage of all total interactions is a measurement that is more comparable as it is bounded between zero and one, whereas the number of implicit interactions can be any natural number, and the prevalence of implicit interactions does not require any additional work to obtain as both the number of implicit interactions and the total number of interactions are available from the implicit interaction identification results [JV17b]. We compare other system properties to both the number of implicit interactions and the percentage of total interactions that are implicit in Section 6.2.

5.1.1 Implicit Interactions in the MCCS

The MCCS described in Section 4.4 has 29 implicit interactions, shown in Table 5.1. These implicit interactions make up 44.62% of all the interactions possible in the system design.

Table 5.1: All 29 implicit interactions that exist within the formal model of the MCCS

ID	Implicit Interaction	ID	Implicit Interaction
x_0	$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H}$	x_{15}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$
x_1	$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$	x_{16}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$
x_2	$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P}$	x_{17}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$
x_3	$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P}$	x_{18}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$
x_4	$\mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$	x_{19}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$
x_5	$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$	x_{20}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C}$
x_6	$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$	x_{21}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$
x_7	$\mathsf{H} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$	x_{22}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$
x_8	$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$	x_{23}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$
x_9	$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{P}$	x_{24}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H}$
x_{10}	$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	x_{25}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{P}$
x_{11}	$\mathsf{H} \rightarrow_{\mathcal{E}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	x_{26}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$
x_{12}	$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	x_{27}	$\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P}$
x_{13}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$	x_{28}	$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P}$
x_{14}	$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H}$		

5.1.2 Implicit Interactions in the WDS

The WDS described in Section 4.5 has 74 implicit interactions, shown in Table 5.2. These implicit interactions make up 52.48% of all the interactions possible in the system.

Table 5.2: All 74 implicit interactions that exist within the formal model of the WDS

ID	Implicit Interaction	ID	Implicit Interaction
x_0	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP1}$	x_{37}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_1	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP1}$	x_{38}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_2	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP1}$	x_{39}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_3	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP2}$	x_{40}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_4	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP2}$	x_{41}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_5	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP2}$	x_{42}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$
x_6	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	x_{43}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$
x_7	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	x_{44}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_8	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	x_{45}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_9	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	x_{46}	$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{10}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	x_{47}	$\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{11}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	x_{48}	$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{12}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	x_{49}	$\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{13}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	x_{50}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{14}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	x_{51}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{15}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	x_{52}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{16}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{53}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{17}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{54}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{18}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{55}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{19}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{56}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$
x_{20}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{57}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$
x_{21}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	x_{58}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{22}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	x_{59}	$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$
x_{23}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	x_{60}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP}$
x_{24}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	x_{61}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP}$
x_{25}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	x_{62}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$
x_{26}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	x_{63}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC}$
x_{27}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	x_{64}	$\mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{28}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	x_{65}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{29}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	x_{66}	$\mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{30}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$	x_{67}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{31}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$	x_{68}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$
x_{32}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	x_{69}	$\mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_{33}	$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	x_{70}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_{34}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{71}	$\mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_{35}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{72}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$
x_{36}	$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	x_{73}	$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$

5.2 Identifying Weak Points

While Table 5.1 and Table 5.2 list all the implicit interactions in the system, even in these relatively small systems it is not immediately apparent what changes we should make to the system to mitigate these identified vulnerabilities. In a large system with hundreds or thousands of implicit interactions, a table such as the ones we provide for the MCCS and WDS is far too much information to comprehend without additional analysis. Given the implicit interactions that exist in a system, we want to identify *weak points* in the system design - parts of the system design that disproportionately contribute to the existence of implicit interactions in the system which we can focus redesign efforts around.

A weak point in a system design can be classified into two types: agent weak points, and interaction weak points. An *agent weak point* refers to an element of $\mathcal{A}$ which contributes to the existence of, or threat posed by, a significant proportion of the implicit interactions in a system. Similarly, an *interaction weak point* refers to an element of $\mathcal{P}_{direct}$ which contributes to the existence of, or threat posed by, a significant proportion of the implicit interactions in a system.

We also consider a special case of interaction weak points called *oversights*, which refers to an element of $\mathcal{P}_{direct}$ that is present within one or more of the implicit interactions in $\mathcal{P}_{implicit}$, but does not occur in any of the intended interactions in $\mathcal{P}_{intended}$. While it is possible to have an agent in a system design which only contributes to the system's implicit interactions, we argue that this is much less likely to occur and

should be more obvious to designers than an interaction oversight. A direct interaction can occur as an oversight more subtly, as it can emerge from the composition of many agents, whereas for an agent to be an oversight, we need to design and introduce a complete agent into a system design which does nothing to contribute to the system's objectives.

5.2.1 Identifying Agent Weak Points

To identify which agents contribute most to the implicit interactions in a system design, we perform a frequency analysis by counting the number of occurrences of each agent in the set of implicit interactions. We also consider how frequently each agent occurs in the system's set of intended interactions, which can give us an idea about which agents contribute most to the system's objectives.

An intended interaction is a walk through the system, so agents are counted once for each occurrence in each interaction of $\mathcal{P}_{intended}$. We do so because agents that occur very frequently in the system's intended interactions may be difficult to redesign without significant changes to the overall system. If we have two agents A and B which contribute to the same number of intended interactions, but A occurs twice in some of those interactions while B does not, then A is more intertwined in the system's operations.

Using both frequency analyses, the *agent contribution ratio* R_{A} for an agent $\mathsf{A} \in \mathcal{A}$ is:

$$R_{\mathsf{A}} = \frac{\sum_{p \in \mathcal{P}_{implicit}} \text{frequency of } \mathsf{A} \text{ in } p}{\sum_{p \in \mathcal{P}_{intended}} \text{frequency of } \mathsf{A} \text{ in } p} \tag{5.2}$$

The agent contribution ratio gives the relative frequency of an agent occurring in the system's implicit interactions to that agent occurring in the system's intended interactions. Agents with a contribution ratio below one occur more frequently in the system's intended interactions than in the system's implicit interactions. They are lower priority because changes to them may inadvertently impact the intended interactions to a greater degree. A value above one means an agent occurs more frequently in the system's implicit interactions than in the system's intended interactions, so changes to these agents have the potential to have a greater impact when reducing the prevalence of implicit interactions.

To determine which agents are weak points, we can select all those agents with a contribution ratio above a certain *ratio threshold* τ defined by the system analyst. Alternatively, we can order all agents by their contribution ratio and select a fixed number of agents with the greatest contribution ratio. Such a decision depends on the available resources for the redesign and mitigation efforts that follow.

5.2.2 Identifying Interaction Weak Points

Identifying interaction weak points is performed in a similar way to identifying agent weak points. We find the frequency that each direct interaction occurs in the set of implicit interactions and intended interactions. Using these frequency analyses, the *interaction contribution ratio* $R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for a direct interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$, $\mathcal{T} \in \{\mathcal{S}, \mathcal{E}\}$ between two agents $\mathsf{A}, \mathsf{B} \in \mathcal{A}$ is:

$$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}} = \frac{\sum_{p \in \mathcal{P}_{implicit}} \text{frequency of } \mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B} \text{ in } p}{\sum_{p \in \mathcal{P}_{intended}} \text{frequency of } \mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B} \text{ in } p} \tag{5.3}$$

The contribution ratio for direct interactions provides a more fine-grained understanding of where issues in the design lie when compared to agent weak points because we consider each communication channel for each agent individually, rather than focusing on agents and their behaviors as an atomic entity. In some cases it can be easier to redesign a single interaction rather than an entire agent, and this additional view of the system can provide a more complete understanding as to why specific agents are identified as agent weak points.

The interaction contribution ratio also provides a means to identify oversights. Any direct interaction that does not occur in the intended interactions will result in a divide-by-zero in the interaction contribution ratio. Therefore, any direct interaction for which the contribution ratio is undefined is an oversight.

As with agent weak points, to determine which interactions are weak points, we can select all those direct interactions with a contribution ratio above a certain analyst-defined ratio threshold τ, or we can select a fixed number of direct interactions with the greatest (or undefined) contribution ratios.

5.2.3 Weak Points in the MCCS

For the MCCS, we will consider the order of agent or direct interaction contribution ratios to indicate the priority for considering parts of the system for redesign. This means that the agent(s) and direct interaction(s) with the highest contribution ratios are the weak points where we should focus redesign efforts.

Table 5.3: Frequency analysis and contribution ratio results for the agents in the MCCS

Agent A	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	R_{A}
S	25	28	0.8929
P	27	32	0.8438
H	26	32	0.8125
C	27	48	0.5625

Table 5.3 shows the results of the frequency analysis and contribution ratio for each agent in the MCCS. We can see that all agents occur with a similar frequency in the system's implicit interactions, but C has by far the lowest of the four contribution ratios due to how intertwined it is with the system's intended interactions. We see that S has the highest contribution ratio despite occurring least frequently in the system's implicit interactions. It also occurs least frequently in the system's intended interactions, so it may be easiest to redesign this agent to reduce the prevalence of implicit interactions in the system design.

Table 5.4 shows the same frequency analysis and contribution ratios for the direct interactions in the MCCS. The results point to the direct interactions $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$ and $\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$ as being the highest priority for redesign efforts. Both direct interactions occur in 1.5 times as many implicit interactions as they do intended interactions. The third highest contribution ratio belongs to the direct interaction $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$, which lends credence to the idea that we should focus design efforts around S, the agent with the highest contribution ratio. Specifically, both direct interaction weak points from S involve `status`, so there may be issues in the design with how this shared environment is used. We also note that all the communication via shared environment

Table 5.4: Frequency analysis and contribution ratio results for the direct interactions in the MCCS

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$
$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$	6	4	1.5
$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$	12	8	1.5
$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$	8	8	1.0
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C}$	7	8	0.875
$\mathsf{H} \rightarrow_{\mathcal{E}} \mathsf{P}$	3	4	0.75
$\mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{P}$	3	4	0.75
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P}$	6	8	0.75
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H}$	5	8	0.625
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	10	16	0.625
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$	6	12	0.5
$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H}$	6	16	0.375
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$	2	12	0.1667
$\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$	2	16	0.125

has contribution ratios in the upper half of the direct interactions when sorted by contribution ratio, with $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$ being the highest possibly due to it being the only direct interaction in either direction between S and P. Figure 5.1 shows the agent and interaction weak points of the MCCS highlighted in red, showing where we should focus redesign efforts.

5.2.4 Weak Points in the WDS

For the WDS, we consider any agent or direct interaction with a contribution ratio greater than two (i.e., $\tau = 2$) to be a weak point, meaning the agent or direct interaction occurs twice as frequently in $\mathcal{P}_{implicit}$ compared to $\mathcal{P}_{intended}$.

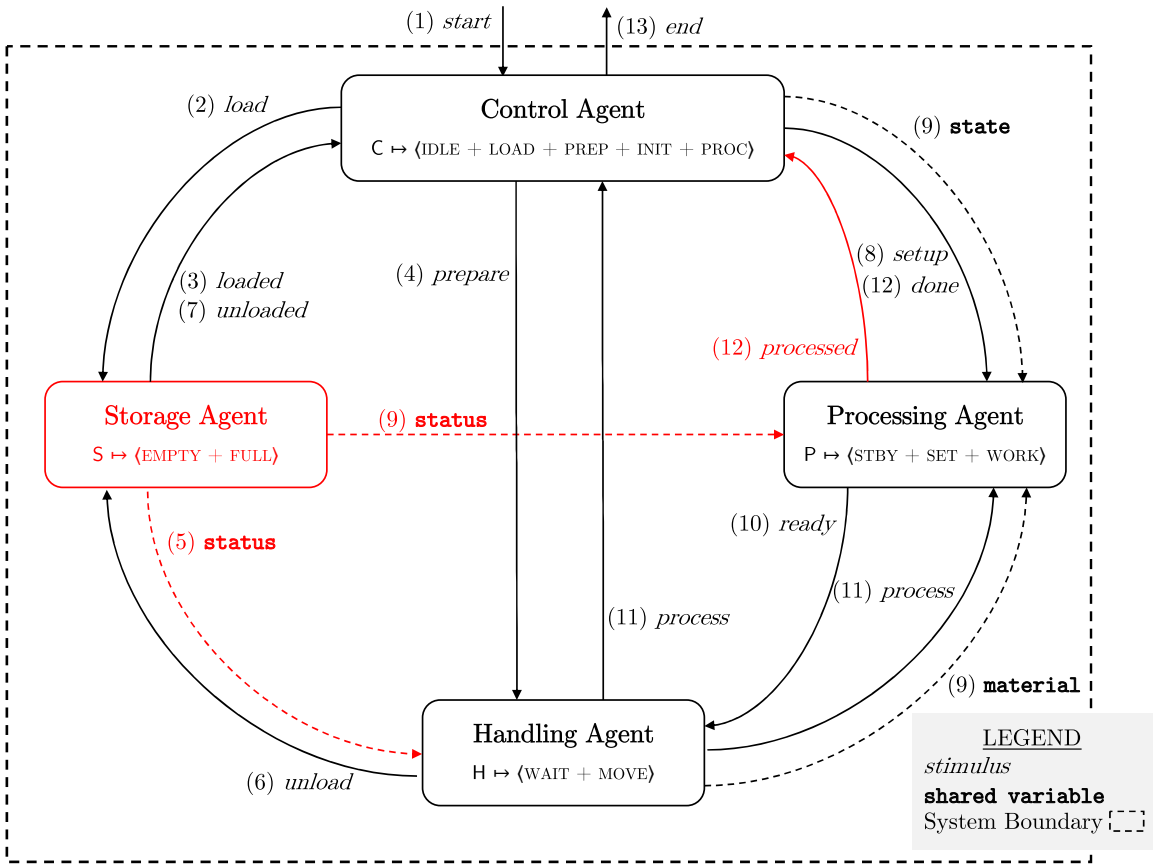


Figure 5.1: Summary of the identified agent and interaction weak points in the MCCS

Table 5.5 shows the results of the frequency analysis and contribution ratio for each agent in the WDS. We can see that PLC, SAP, and SO3 all occur most frequently in the WDS implicit interactions. However, when we also consider the system's intended interactions, we can see that OP has a higher contribution ratio than any of these agents. This points to a problem with how OP communicates with the rest of the system that warrants further investigation. In the WDS, OP is responsible for fixing faults and broadcasting stimuli to other agents once the error state has been resolved, so the level of granularity that the agent contribution ratios provides tells us that either the issue lies in at least one of the fault-reporting or fault-fixing functionalities.

Table 5.5: Frequency analysis and contribution ratio results for the agents in the WDS

Agent A	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	R_{A}
OP	54	18	3.0
SO3	62	25	2.48
SFM	32	16	2.0
SAP	62	45	1.3778
PLC	65	54	1.2037
CFP1	3	9	0.3333
CFP2	3	9	0.3333

We see that R_{SO3} also ranks highly, while agents SAP and PLC have much lower contribution ratios despite having a similar frequency in $\mathcal{P}_{implicit}$. From this analysis and our defined ratio threshold, we identify OP, SO3, and SFM as agent weak points and prioritize them for redesign in that order.

Table 5.6 shows the same frequency analysis and contribution ratios for the direct interactions in the WDS. Immediately, these results point to an issue in the design that was not apparent in the agent weak point analysis. We can see that three of the direct interactions that exist in the system, all among the most frequently occurring in the system's implicit interactions, do not occur in the system's intended interactions at all. These three direct interactions, $\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$, $\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$, and $\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$, are oversights. It is also possible to visually identify oversights by the existence of any direct interactions appearing in $\mathcal{P}_{implicit}$, as can be seen with x_{16}, x_{27}, and x_{68} in Table 5.2, however, in a system with hundreds or thousands of implicit interactions, this is not a practical process.

Table 5.6: Frequency analysis and contribution ratio results for the direct interactions in the WDS

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$
$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	23	0	–
$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	22	0	–
$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	18	0	–
$\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	22	8	2.75
$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	17	8	2.125
$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	17	9	1.8889
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	7	4	1.75
$\mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	11	8	1.375
$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	11	8	1.375
$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP}$	20	18	1.1111
$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	10	9	1.1111
$\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$	8	8	1.0
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	8	8	1.0
$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	7	18	0.3889
$\mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP1}$	3	9	0.3333
$\mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP2}$	3	9	0.3333
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SFM}$	0	8	0
$\mathsf{SFM} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	0	8	0

These oversights offer a useful explanation as to why OP is ranked highest in Table 5.5, as two of the three oversights have OP as the source agent. As oversights do not contribute to the system's objectives, we should aim to eliminate them from the system first. Along with these oversights, $\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$ and $\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC}$ have contribution ratios above the defined ratio threshold ($\tau = 2$), and are therefore interaction weak points, but changes to first eliminate the oversights currently in the system may result in these direct interactions no longer being relevant.

The weak points identified in the WDS are highlighted in Fig. 5.2. Identified agent weak points OP, SO3, and SFM, and interaction weak points $\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$ and

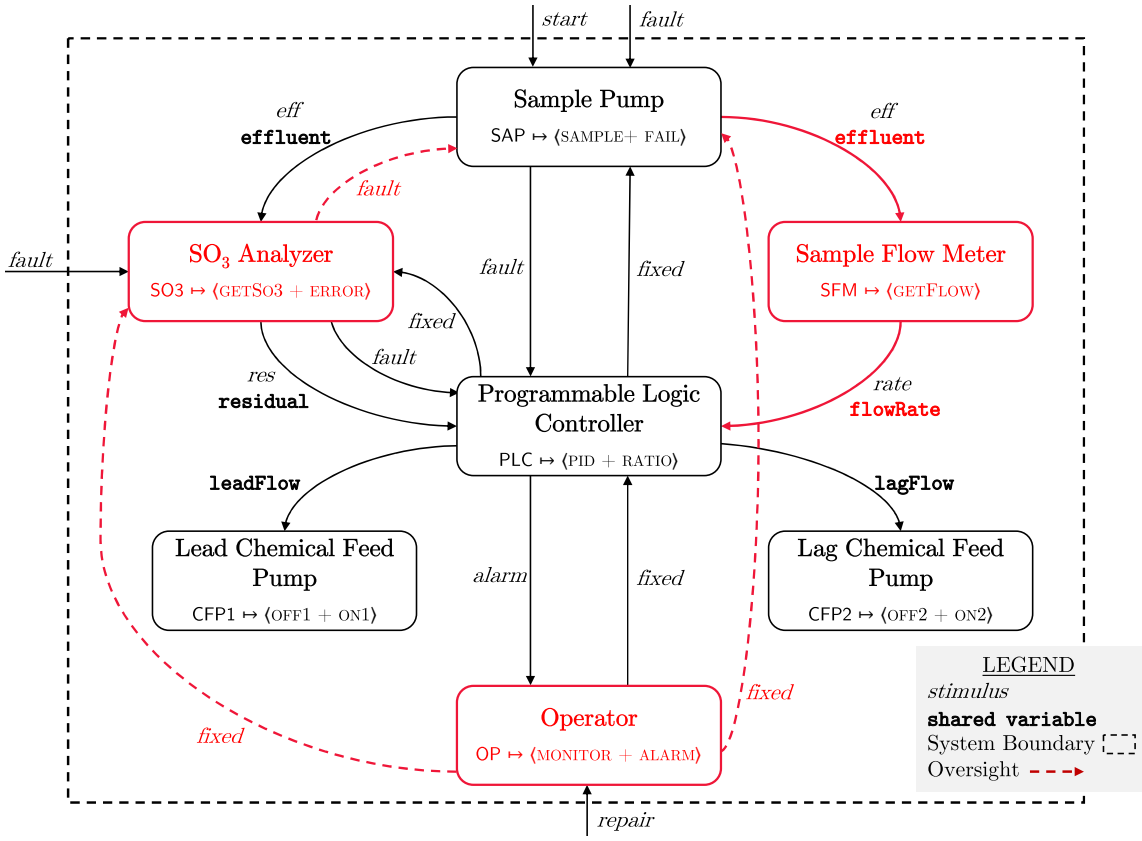


Figure 5.2: Summary of the identified agent and interaction weak points in the WDS

SFM $\rightarrow_{\mathcal{E}}$ PLC are highlighted in red, with the oversights OP $\rightarrow_{\mathcal{S}}$ SAP, OP $\rightarrow_{\mathcal{S}}$ SO3, and SO3 $\rightarrow_{\mathcal{S}}$ SAP added as red dashed arrows.

From Fig. 5.2, we see the agent weak points and interaction weak points consistently point to problematic areas of the system design as each interaction weak point has an agent weak point either as the source or sink agent. We can now understand why OP and SO3 are agent weak points: both are sources of oversights. SFM is not involved in any oversights, but the interaction weak points help to understand the issue with this agent. The interaction weak points show that while SFM takes part in four different direct interactions, the issue lies solely in the agent's two communications

via shared environments. This helps to narrow where to focus efforts as we can see in Table 5.6 that the agent's communication via stimuli does not contribute to the system's implicit interactions.

Intuitively, we might expect PLC to be an agent weak point since it is highly centralized in the design, and its influence can spread widely in the system. However, due to its high frequency in $\mathcal{P}_{intended}$, PLC has a relatively low contribution ratio. PLC is so intertwined with the system's intended interactions that redesign efforts can be better spent on other agents. Furthermore, because the agent weak points all communicate directly with PLC, mitigating the implicit interactions the weak points contribute to could, by extension, reduce this agent's contribution to implicit interactions.

5.3 Generating Design Changes

To generate design alternatives, we use the identified weak points to target specific model transformations. The changes are not necessarily only to the specific agent or direct interaction identified by a weak point; the weak points serve as a means to guide the thought process towards a design change rooted in the issue that a weak point helps to highlight. The design changes made are transformations to the formal models in Section 4.4 and Section 4.5. Rather than provide the full formal specification for every design alternative created, we describe each model transformation, labelled in bold (e.g., **T1**). A design alternative is a set of model transformations applied to the original system design, and are labelled as **MCCSx** and **WDSx** where $x \in \mathbb{N}$ (e.g., **MCCS1** is the original design alternative of the MCCS, and **WDS1** is

the original for design alternatives of the WDS). We present new versions of the original collaboration diagram for the MCCS (Figure 4.1), the original communication diagram for the WDS (Figure 4.5), and intended interactions where appropriate.

5.3.1 Design Alternatives for the MCCS

The majority of the model transformations for the MCCS are directly identified by analysis of the weak points identified in system design alternatives. This provides an examples of the formal methods-based iterative design process discussed in Section 3.5. Additional model transformations are identified after several rounds of identifying weak points and new design alternatives to further experiment with various system properties.

Addressing the Weak Points in the Original System

From the interaction weak points $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$, $\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$, and $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$, we choose to focus on the shared variable `status` as this is the variable that enables two of the three weak points. The purpose of `status` in the system is for P and H to be able to check that S is in the correct state when the C issues commands to them. In a distributed system with concurrent operations, this synchronization is useful to improve quality attributes such as reliability, however, from purely the viewpoint of implicit interactions, we want to remove this capability.

If C is meant to control the system operations, as its name and most of its actions suggest, one possible change is for P and H to simply trust that C only issues commands

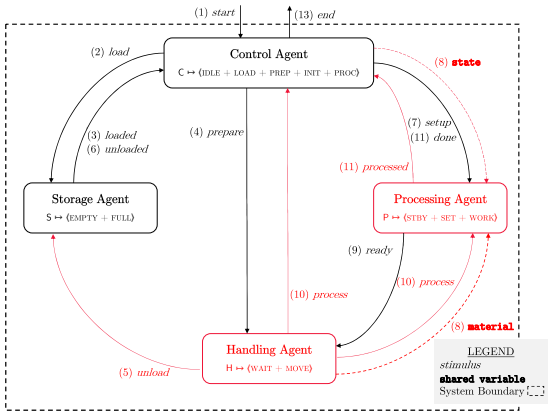
to them when S is in the correct state. Another alternative is to retain a shared environment for `status`, but move the variable definition to C so we no longer have communication via shared environments from S to other agents. These alternatives are given in the following transformations:

T1 Remove the capability for P and H to reference `status` in their concrete behaviors.

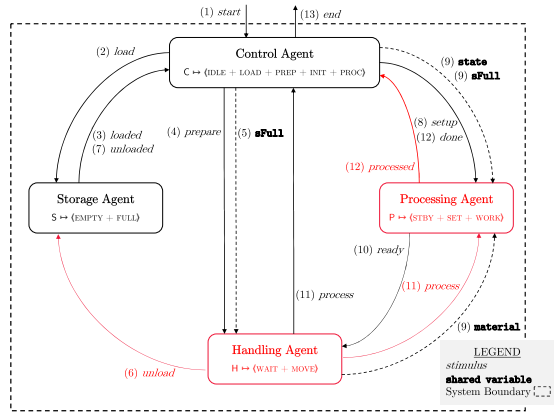
T2 Rename `status` to `sFull`. Rather than defining the environment in S, set `sFull` to 1 when C receives *loaded* and set `sFull` to 0 when C receives *unloaded*.

These changes are given in Figure 5.3. For every design alternative we produce, we perform the formal analysis activities in Section 5.1 and Section 5.2 to identify implicit interactions and weak points that can be compared across system designs. For example, Figure 5.3a shows the results of applying transformation T1, resulting in a new system design **MCCS2** that now has 15 implicit interactions, down from 29 in the original system design, and making up 35.71% of the total possible interactions in the system, down from 44.62% in the original system design. Figure 5.3b shows the results of applying transformation T2, resulting in another new system design **MCCS3** that now has 19 implicit interactions, making up 37.25% of the total possible interactions in the system.

The weak points in these system design alternatives also change and are highlighted in red in Figure 5.3a and Figure 5.3b. Having completely eliminated the direct interactions $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$ and $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$, neither these interactions, nor agent S, are weak points. Despite the change, $\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$ is still an interaction weak point in both new design



(a) Alternative design **MCCS2** with weak points eliminated by removing `status` references, after applying transformation **T1** to **MCCS1**



(b) Alternative design **MCCS3** with weak points eliminated by moving `status` definition, after applying transformation **T2** to **MCCS1**

Figure 5.3: Alternative designs to eliminate weak points $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$ and $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$ in the MCCS

alternatives, as are a number of new agents and direct interactions. The interaction contribution ratios for each design alternative, along with the contribution ratios of the original system design MCCS1, are shown in Table 5.7 to show how the small changes in transformations T1 and T2 impact the contribution ratios. In Table 5.7, "n/a" indicates that the direct interaction does not occur in either the implicit or intended interactions of the system design.

Centralizing the System to Eliminate New Weak Points

In both new designs, agents H and P are the two agents with the highest contribution ratios. In each, communication between those two agents are the interaction weak points, as is the direct interaction $\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$, which is the only way for S and H to directly communicate in either system design. Based on this, we aim to make

Table 5.7: Interaction contribution ratios for the original MCCS side-by-side with the contribution ratios for the alternative designs generated by transformations **T1** and **T2**

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for **MCCS1**	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for **MCCS2**	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for **MCCS3**
$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$	1.5	n/a	n/a
$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$	1.5	2.0	1.3333
$\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$	1.0	n/a	n/a
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C}$	0.875	1.0	0.5
$\mathsf{H} \rightarrow_{\mathcal{E}} \mathsf{P}$	0.75	1.0	0.5
$\mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{P}$	0.75	1.0	0.5
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{P}$	0.75	1.0	0.8333
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H}$	0.625	0.6667	0.6667
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	0.625	0.8333	0.4167
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{S}$	0.5	1.25	0.75
$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{H}$	0.375	0.8333	0.4167
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$	0.1667	0.2	0.1
$\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$	0.125	0.2	0.15
$\mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{H}$	n/a	n/a	0.6667

another round of design changes that focus on the weak points between non-control agents and build on the idea that C should provide more complete control as its name suggests. We aim to fully centralize the system around C with transformation T3, which we do in an incremental fashion to observe how the prevalence of implicit interactions changes in the intermediate step:

T3 Completely centralize agent behavior coordination in the control agent C

T3a Move the remaining shared variable `material` definition from H to C so all shared environment communication is routed through C. Rename the variable to `hHasMaterial` which is set to 1 when C receives *unloaded* and set to 0 when C receives *processed*

T3b Reroute all communication via stimuli through C

Each step for transformation T3 is applied to MCCS3 to produce design alternative **MCCS4**, shown in Figure 5.4. With T2, T3a, and T3b applied to MCCS1 to produce **MCCS5**, communication in the system to achieve its objectives is now structured as shown in Figure 5.5. The system still performs its fundamental actions in the same order: Once C receives *start*, S loads material, then H prepares to receive material, material in S is unloaded to H, P is setup, P processes the material, and H and P are reset. The difference in this system design is that C now has total control over the sequence of operations.

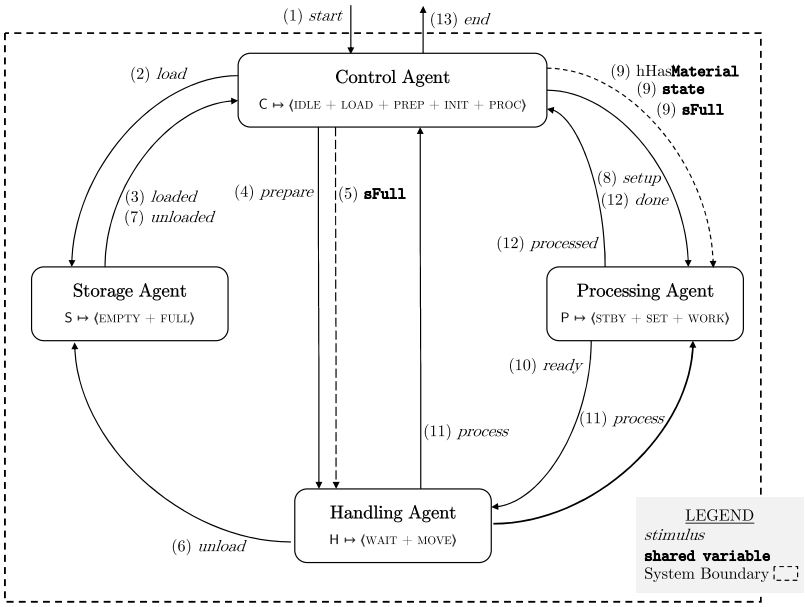


Figure 5.4: Alternative design **MCCS4** with shared environments centralized around C, after applying transformations **T2** and **T3a** to **MCCS1**

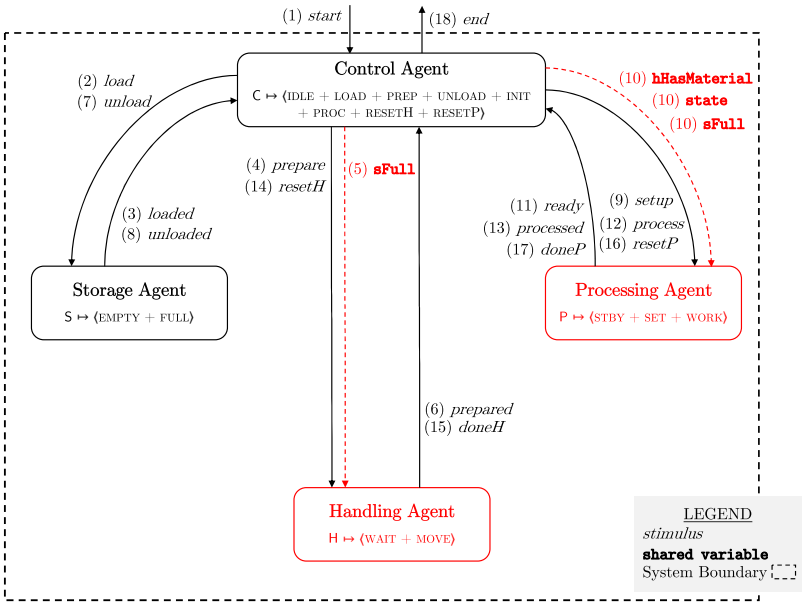


Figure 5.5: Alternative design MCCS5 with control completely centralized around C, after applying transformations T2, T3a, and T3b to MCCS1

Eliminating Unnecessary Communication to Simplify System Design

Again, the interaction contribution ratios for MCCS5 are given in Table 5.8 and weak points are shown in red on Figure 5.5. The weak points for this highly centralized system all point towards the communication via shared environment being an issue. The shared environments once served to validate the state of other agents, but as C now has tight control over the other agents, we can eliminate the communication via shared environment and convert the MCCS to a fully message-passing system with transformation T4:

T4 Remove the references to shared variables `sFull`, `hHasMaterial`, and `state` in P, and `sFull` in H.

Table 5.8: Frequency analysis and contribution ratio results for the direct interactions in the centralized manufacturing control cell (**MCCS5**)

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$
$\mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{H}$	1	4	0.25
$\mathsf{C} \rightarrow_{\mathcal{E}} \mathsf{P}$	1	8	0.125
$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$	2	24	0.08333
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C}$	1	16	0.0625
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	1	16	0.0625
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H}$	0	12	0
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$	0	16	0
$\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$	0	16	0

Applying this transformation to MCCS5 produces a new alternative system design **MCCS6** as shown in Figure 5.6, which now has almost no implicit interactions and the majority of the contribution ratios, shown in Table 5.9, are zero. This means that most parts of the system only contribute to its intended interactions, which provides a good stopping point for these iterative changes.

Table 5.9: Frequency analysis and contribution ratio results for the direct interactions in the message-passing centralized MCCS (**MCCS6**)

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{S}$	1	2	0.5
$\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$	1	3	0.3333
$\mathsf{H} \rightarrow_{\mathcal{S}} \mathsf{C}$	0	2	0
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$	0	3	0
$\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{H}$	0	2	0
$\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$	0	2	0

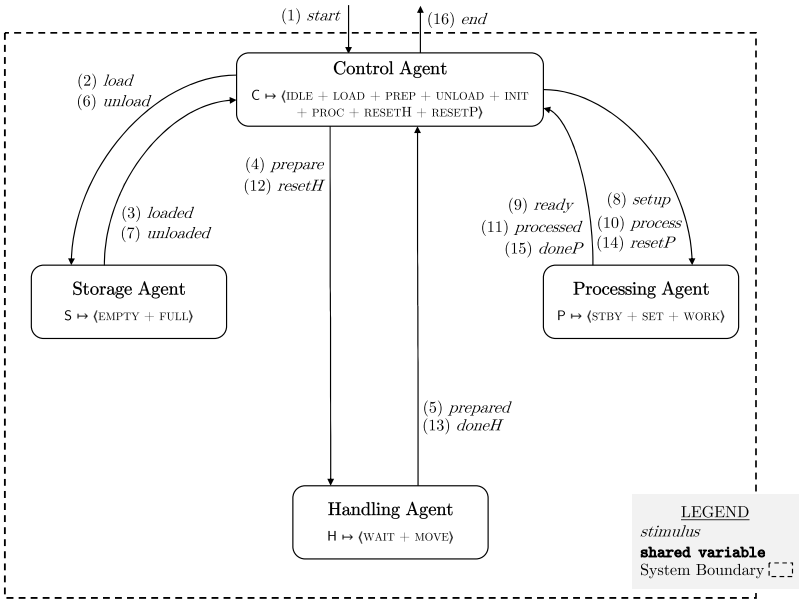


Figure 5.6: Alternative design MCCS6 with only message passing and control completely centralized around C, after applying transformation T4 to MCCS5

Additional Manufacturing Cell Control System Design Alternatives

The iterative application of design changes and weak point identification above has led to a highly centralized design, but there is another option. The purpose of C is to control operations, but in the original system design it already does not perform this responsibility by itself. Therefore, we can also decentralize the system by removing the C entirely and coordinating the remaining three agents through a more distributed control. With a more distributed approach, we can also then eliminate some of the weak points in the original system: If S directly communicates with P when material is unloaded, then P does not need to check `status` anymore, so we can eliminate $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$, and $\mathsf{P} \rightarrow_{\mathcal{S}} \mathsf{C}$ no longer exists because C is not in the system. We make these changes in incremental steps according to the following transformations:

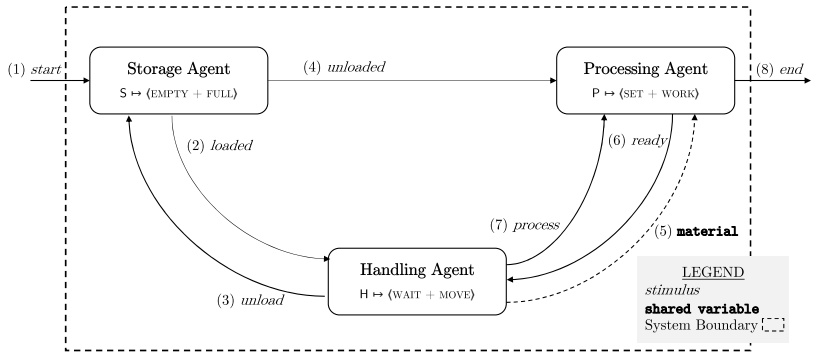


Figure 5.7: Alternative design **MCCS8** with control distributed by removing the control agent C, after applying transformations **T5a** and **T5b** to **MCCS1**

T5 Distribute control across all agents by removing C from the system entirely.

- **T5a** Remove C from the system, restructuring communication via stimuli to coordinate the remaining operations without a centralized control agent.
- **T5b** Remove shared variable `status` as it is no longer needed since S communicates directly via stimuli with P

Starting from the original system design MCCS1, applying transformation T5a results in **MCCS7**, and applying both transformations T5a and T5b results in **MCCS8**. The communications in MCCS8 to achieve its objectives are structured as shown in Figure 5.7. Without the overhead of the additional control agent, *start* immediately triggers S to load material, which is then unloaded by H, triggering P to setup and then process the material.

Finally, an interesting property of centralizing the system over multiple iterations of weak point identification and redesign is how there is very little concurrency in the system designs in Figure 5.5 and Figure 5.6. We can introduce some more concurrency to the system by changing how the system resets H and P. Instead of resetting H first

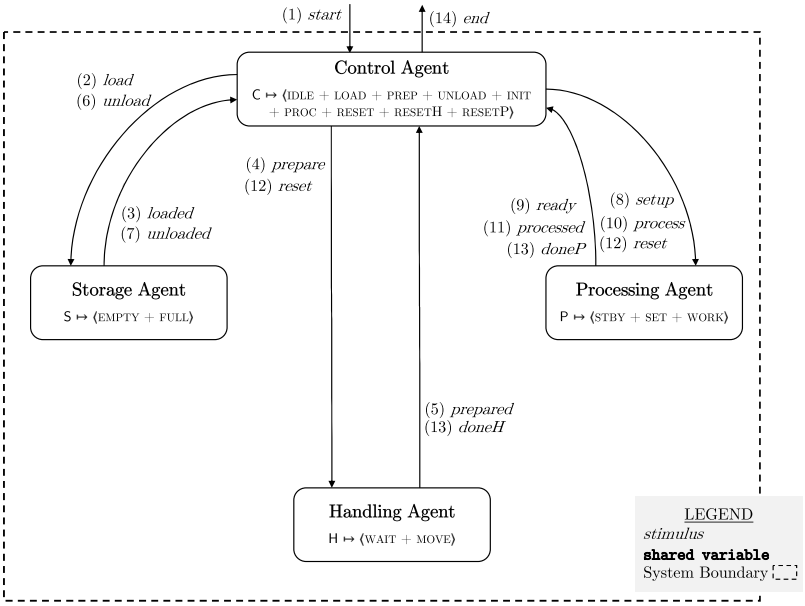


Figure 5.8: Alternative design **MCCS10** in which **T6** is applied to design **MCCS6** to simultaneously reset H and P

and P second, we can broadcast a reset stimulus to both to reset them for the next round of operations at the same time:

T6 Replace *resetH* and *resetP* with a single stimulus *reset*. Have C only send *end* once both *doneH* and *doneP* are received in either order.

By applying T6 to MCCS5 in Figure 5.5 and to MCCS6 in Figure 5.6, we produce design alternatives **MCCS9** and **MCCS10** respectively. The collaboration diagram for MCCS10 is shown in Fig. 5.8. The structure of the collaboration diagram is overall very similar to that of Fig. 5.6, but the change described in T6 results in a slightly higher prevalence of implicit interactions. A similar change in the prevalence of implicit interactions occurs between MCCS5 and MCCS9 from the added concurrency introduced by T6.

Implicit Interaction Dataset for the MCCS Design Alternatives

The above transformations are combined to generate a total of 10 formal models for alternative designs of the MCCS. Table 5.10 shows the prevalence of implicit interactions in each of these of these system design alternatives.

Table 5.10: Measurements of the prevalence of implicit interactions for each system design alternative for the MCCS

System Design Alternative	Transformations Applied	Number of Implicit Interactions	Prevalence of Implicit Interactions
MCCS1	-	29	44.61%
MCCS2	**T1**	15	35.71%
MCCS3	**T2**	19	37.25%
MCCS4	**T2**, **T3a**	17	38.64%
MCCS5	**T2**, **T3a**, **T3b**	3	16.67%
MCCS6	**T2**, **T3a**, **T3b**, **T4**	1	8.333%
MCCS7	**T5a**	3	17.65%
MCCS8	**T5a**, **T5b**	2	18.18%
MCCS9	**T2**, **T3a**, **T3b**, **T6**	4	22.22%
MCCS10	**T2**, **T3a**, **T3b**, **T4**, **T6**	2	16.67%

MCCS1 is the original system design, MCCS2 through MCCS6 are generated solely from identifying weak points to inform design changes, and the remaining MCCS7 through MCCS10 are alternative system designs created as opposites to some of the side effects that the weak-point-focused design changes produced. We see that overall, the original system design MCCS1 has both the highest number of implicit interactions and the highest prevalence of implicit interactions, which makes sense as most of the design alternatives generated are created by applying the methods for identifying weak points. MCCS6, the design alternative produced at the end of the

iterative formal analysis process, has the lowest prevalence of implicit interactions overall.

5.3.2 Design Alternatives for the WDS

While some of the model transformations for the WDS are based solely in weak point identification, this larger system presents an opportunity to see how different independent design considerations might impact the prevalence of implicit interactions when considered in isolation, and together. Model transformations are also created to address design considerations such as centralization and concurrency, which are combined whenever possible to produce many more design alternatives than the MCCS.

Eliminating Design Oversights

The way in which faults are modelled in the WDS is the issue that causes the oversight $\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$. While there is nothing wrong with modelling a fault as an external stimulus, either SO3 or SAP receiving a fault causes it to broadcast another *fault*, which will cause the other agent to trigger its own faulting actions. This is because communication via stimuli in C^2KA is broadcast, so any stimuli propagation such as this needs to be handled carefully. We need separate stimuli to model each fault, as described in the following model transformation:

TI Instead of both SO3 and SAP sending *fault* when a fault occurs, SO3 sends *so3fault* and SAP sends *sapfault*

The oversights OP $\rightarrow_{\mathcal{S}}$ SAP and OP $\rightarrow_{\mathcal{S}}$ SO3 both have the same root cause in the model. Like the *fault* stimuli propagated by SO3 and SAP, the WDS intends to have PLC propagate a *fixed* stimulus to SAP and SO3 when it receives *fixed* from OP. However, since both stimuli have the same name, OP broadcasting *fixed* actually influences all of PLC, SAP, and SO3, and PLC broadcasting *fixed* influences SAP and SO3 a second time. We can therefore change the system in one of two ways. We need to modify the system design by either renaming appropriate stimuli to fix the intended stimuli propagation in the original design, or we must do away with stimuli propagation and directly communicate the *fixed* stimuli.

TII Rename the stimulus that PLC sends in response to receiving *fixed* to *fixed2*

TIII PLC should not propagate the *fixed* stimulus at all. Instead, OP will directly send *fixed* to PLC, SAP, and SO3. PLC will no longer send any stimulus in response to receiving *fixed*.

For consistency, we will also broadcast the fault signals from SAP and SO3 directly to PLC and OP which we will perform in multiple incremental steps to observe what, if any, impact an inconsistent design has on the prevalence of implicit interactions in that system:

TIV Send fault signals from SAP and SO3 directly to PLC and OP instead of sending fault signals to PLC and PLC broadcasting the stimulus *alarm*

TIVa Broadcast only the fault signal from SAP directly to PLC to OP. The fault signal from SO3 is still propagated through PLC to OP and the fixed signal from OP is still propagated through PLC.

TIVb Broadcast both fault signals from SAP and SO3 directly to PLC to OP.

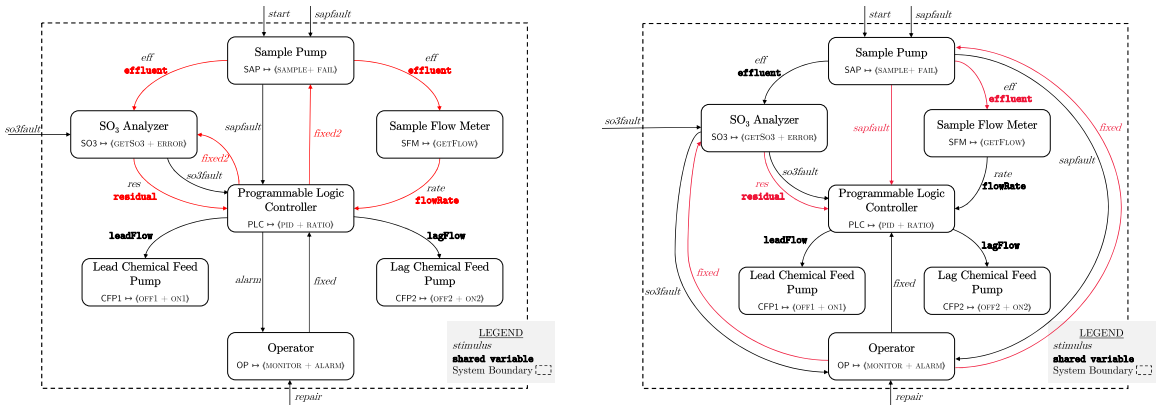
From these transformations we create multiple design alternatives, varying which oversights are still present in the system to understand how different oversights might impact a system differently. Table 5.11 summarizes all the design alternatives generated from the above model transformations, and which oversights are still present in each alternate design.

Table 5.11: WDS design alternatives with varying numbers of oversights present in the system

System Design Alternative	Transformations Applied	Oversight SO3 $\rightarrow_{\mathcal{S}}$ SAP Present?	Oversight OP $\rightarrow_{\mathcal{S}}$ SAP Present?	Oversight OP $\rightarrow_{\mathcal{S}}$ SO3 Present?
WDS1	-	✓	✓	✓
WDS2	**TI**	✗	✓	✓
WDS3	**TII**	✓	✗	✗
WDS4	**TI**, **TII**	✗	✗	✗
WDS5	**TIII**	✓	✗	✗
WDS6	**TIVa**	✓	✓	✓
WDS7	**TIVb**	✓	✓	✓
WDS8	**TI**, **TIVa**	✗	✓	✓
WDS9	**TI**, **TIVb**	✗	✓	✓
WDS10	**TI**, **TIII**, **TIVa**	✗	✗	✗
WDS11	**TI**, **TIII**, **TIVb**	✗	✗	✗

WDS4 and WDS11 are the two alternative designs from Table 5.11 in which we are primarily interested, as these are the end results of applying transformations to remove all oversights. WDS10 also has no oversights, but this design alternative is only a stepping stone towards WDS11 that we include to have a more fine-grained view on the model transformations we apply. The elimination of oversights in either WDS4 or WDS11 greatly reduces the prevalence of implicit interactions in the system. While the original system has 74 implicit interactions, making up 52.48% of all possible

interactions, both the design alternatives have only 15 implicit interactions, with those 15 interactions making up 28.30% of all interactions in the design alternative WDS4, shown in Figure 5.9a, and 19.48% of all interactions in the design alternative WDS11, shown in Figure 5.9b.



(a) Alternative design **WDS4** to fix the model's intended stimuli propagation, after applying transformations **TI** and **TII** to **WDS1**

(b) Alternative design **WDS11** that directly communicates all *fixed* and *fault* stimuli, after applying transformations **TI**, **TIII**, **TIVb** to **WDS1**

Figure 5.9: Alternative designs to fully eliminate oversights in the WDS

While the number of implicit interactions in both design alternatives is similar, the new weak point show greater difference. The interaction contribution ratios for the design alternatives in Figure 5.9 are shown in Table 5.12 along with those for the original system for comparison. In Table 5.12, "n/a" indicates that the direct interaction does not occur in either the implicit or intended interactions of the system design, while "–" indicates that the direct interaction is an oversight. From Table 5.12, we can see that by renaming the stimuli to remove oversights, those communications channels that were the oversights no longer exist, whereas renaming oversights by broadcasting stimuli directly between OP and other agents results in new communication channels not present in either other design alternative. The design alternatives in Figure 5.9

have the six direct interactions with the largest contribution ratios highlighted in red. In Figure 5.9b, the two highest interaction contribution ratios are for those direct interactions that were oversights in the original design. This points towards the design alternative in Figure 5.9a perhaps being a better solution to eliminate those oversights.

Table 5.12: Interaction contribution ratios for the original WDS side-by-side with the contribution ratios for the alternative designs generated by transformations in Figure 5.9a and Figure 5.9b

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for **WDS1**	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for **WDS4**	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$ for **WDS11**
$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	–	n/a	3.0
$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	–	n/a	2.0
$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	–	n/a	n/a
$\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SFM}$	2.75	0.625	1.0
$\mathsf{SFM} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	2.125	0.5	0.5
$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SAP}$	1.8889	1.0	n/a
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	1.75	0.25	1.0
$\mathsf{SO3} \rightarrow_{\mathcal{E}} \mathsf{PLC}$	1.375	0.25	1.0
$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	1.375	0.1667	1.0
$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{OP}$	1.1111	0	n/a
$\mathsf{PLC} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	1.1111	0.3	n/a
$\mathsf{SAP} \rightarrow_{\mathcal{E}} \mathsf{SO3}$	1.0	0.375	0.375
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SO3}$	1.0	0	0
$\mathsf{OP} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	0.3889	0.2	0
$\mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP1}$	0.3333	0.1	0.1667
$\mathsf{PLC} \rightarrow_{\mathcal{E}} \mathsf{CFP2}$	0.3333	0.1	0.1667
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{SFM}$	0	0	0
$\mathsf{SFM} \rightarrow_{\mathcal{S}} \mathsf{PLC}$	0	0	0
$\mathsf{SO3} \rightarrow_{\mathcal{S}} \mathsf{OP}$	n/a	n/a	0.375
$\mathsf{SAP} \rightarrow_{\mathcal{S}} \mathsf{OP}$	n/a	n/a	0

Introducing an Additional Agent to Distribute Control

It is interesting that in Figure 5.9a, those direct interactions that are highlighted in red form cycles made up of a combination of direct interactions that are part of the chemical pump-controlling process of the system and fault-managing process. PLC is central to these two separate processes that occur in the system and the question becomes whether one agent should control two separate system processes, as this could serve as a point where the potential communication paths can "jump" from one intended control flow to the other and lead to a higher prevalence of implicit interactions. We experiment with adding a fault manager agent called FAULTMNGR that can help to decouple each process by coordinating the communications related to reporting and fixing faults while PLC continues to coordinate the communications related to controlling the chemical feed pumps:

TV Introduce agent FAULTMNGR. When a fault occurs in SAP or SO3, they send a signal to FAULTMNGR instead of PLC, and FAULTMNGR sends *alarm* to PLC and OP. When the fault is resolved, OP sends a fixed signal to FAULTMNGR, which then broadcasts a signal to PLC, SAP, and SO3.

Figure 5.10 shows the application of model transformation TV to the design alternative in Figure 5.9a to produce **WDS13**. Another version of the system is also created, **WDS12**, which adds the FAULTMNGR into the original system design with oversights still present. In WDS13, there are 23 implicit interactions making up 31.08% of all possible interactions in the system, an increase from the system without the fault manager agent. The weak points highlighted in red are those direct interactions that

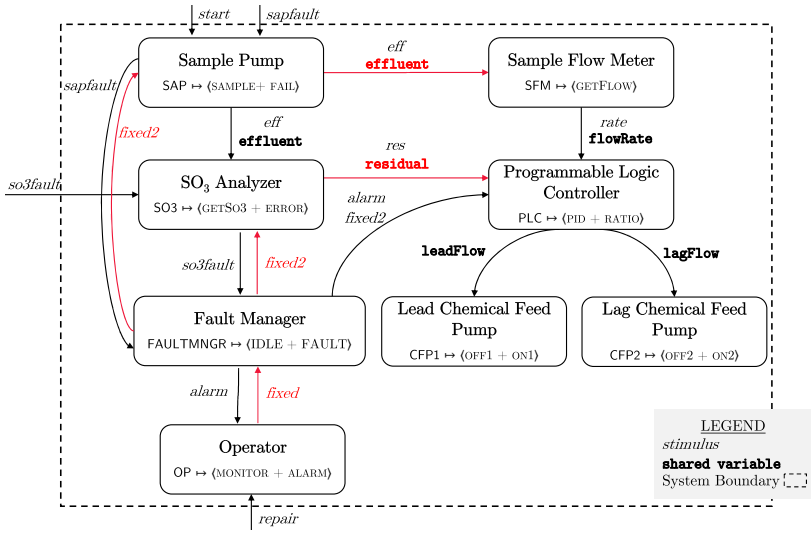


Figure 5.10: Alternative design **WDS13** with FAULTMNGR introduced, after applying transformations **TI**, **TII**, and **TV** to **WDS1**

still have a contribution ratio above one in Table 5.13. When we look at all these interaction weak points together, we see in Figure 5.10 that the weak points form paths made up of direct interactions from the two separate processes in the WDS. The paths in red in Figure 5.10 flow from OP and FAULTMNGR which only take part in one of the two system processes, through SAP and SO3 which take part in both processes in the system, to PLC and SFM which only take part in the other system process. This issue is reminiscent of the idea of non-interference, in which a system's low-level inputs should produce the same results regardless of the system's high-level inputs [GM82], and warrants further investigation.

The Role of Interference in the Prevalence of Implicit Interactions

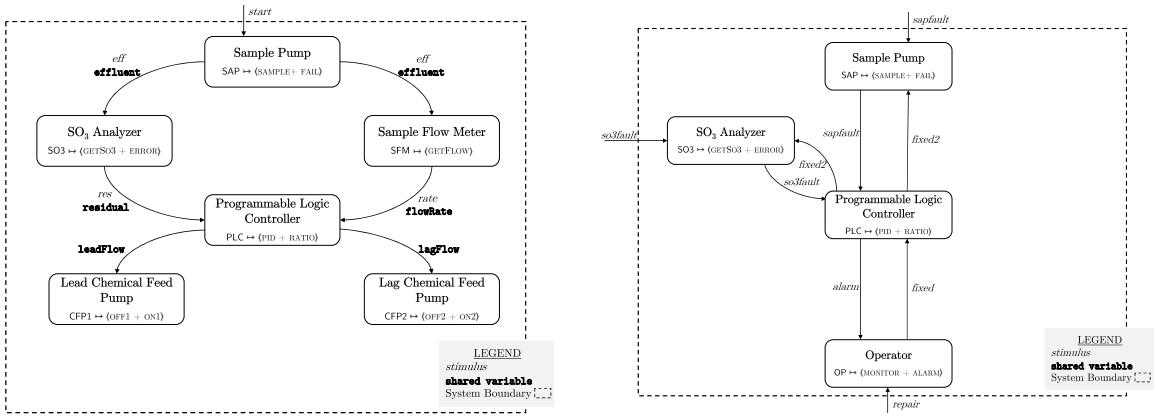
To investigate how multiple processes in one system may be interfering to produce more implicit interactions, we use the WDS design alternative WDS4, which has no

Table 5.13: Frequency analysis and contribution ratio results for the direct interactions in the WDS with a fault manager agent (Transformations **TI**, **TII**, and **TV**)

Direct Interaction $\mathsf{A} \rightarrow_{\mathcal{T}} \mathsf{B}$	Sum Frequency in $\mathcal{P}_{implicit}$	Sum Frequency in $\mathcal{P}_{intended}$	$R_{\mathsf{A}\rightarrow_{\mathcal{T}}\mathsf{B}}$
FAULTMNGR $\rightarrow_{\mathcal{S}}$ SAP	11	3	3.6667
FAULTMNGR $\rightarrow_{\mathcal{S}}$ SO3	10	3	3.3333
SO3 $\rightarrow_{\mathcal{E}}$ PLC	11	4	2.75
SAP $\rightarrow_{\mathcal{E}}$ SFM	6	4	1.5
OP $\rightarrow_{\mathcal{S}}$ FAULTMNGR	9	9	1.0
SAP $\rightarrow_{\mathcal{S}}$ FAULTMNGR	4	5	0.8
SFM $\rightarrow_{\mathcal{E}}$ PLC	3	4	0.75
SAP $\rightarrow_{\mathcal{E}}$ SO3	4	9	0.4444
PLC $\rightarrow_{\mathcal{E}}$ CFP1	4	11	0.3636
PLC $\rightarrow_{\mathcal{E}}$ CFP2	4	11	0.3636
SO3 $\rightarrow_{\mathcal{S}}$ FAULTMNGR	3	10	0.3
FAULTMNGR $\rightarrow_{\mathcal{S}}$ PLC	2	9	0.2222
SAP $\rightarrow_{\mathcal{S}}$ SFM	0	4	0
FAULTMNGR $\rightarrow_{\mathcal{S}}$ OP	0	9	0
SAP $\rightarrow_{\mathcal{S}}$ SO3	0	9	0
SFM $\rightarrow_{\mathcal{S}}$ PLC	0	4	0
SO3 $\rightarrow_{\mathcal{S}}$ PLC	0	4	0

oversights and is shown in Figure 5.9a. We create models that capture just one of two processes in the system. Then, we create four models that each include only the communication that propagates from one of the four stimuli that enter the system from outside the system boundary.

We know that the complete design in Figure 5.9a has 15 implicit interactions, but when we separate the system into only the communication to control chemical feed pumps in Figure 5.11a and only the communication to report and fix faults in Figure 5.11b, we find many fewer. The model in Figure 5.11b has only two implicit interactions, SAP $\rightarrow_{\mathcal{S}}$ PLC $\rightarrow_{\mathcal{S}}$ SO3 and SO3 $\rightarrow_{\mathcal{S}}$ PLC $\rightarrow_{\mathcal{S}}$ SAP, while the model in



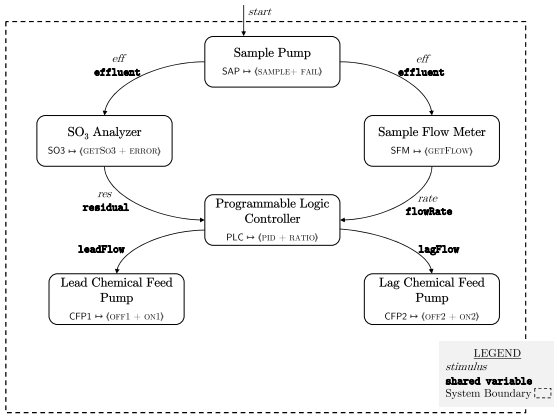
(a) Only communication that makes up the sampling and pump control process

(b) Only communication that makes up the fault reporting and fixing process

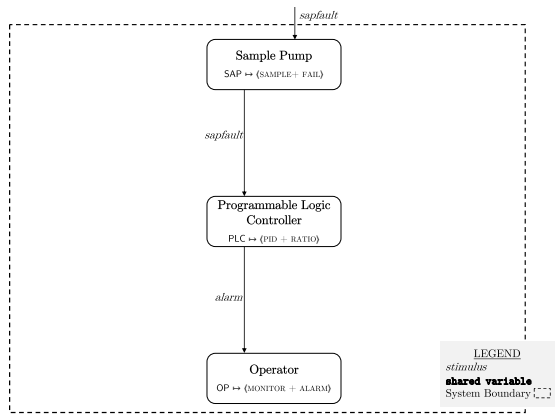
Figure 5.11: Alternative designs of the WDS that capture only one of two processes in the system

Figure 5.11a has no implicit interactions. The remaining 13 implicit interactions in WDS4 come from a combination of communications from the two processes. This is an aspect of system designs not seen in the MCCS, as it only contains one main control flow.

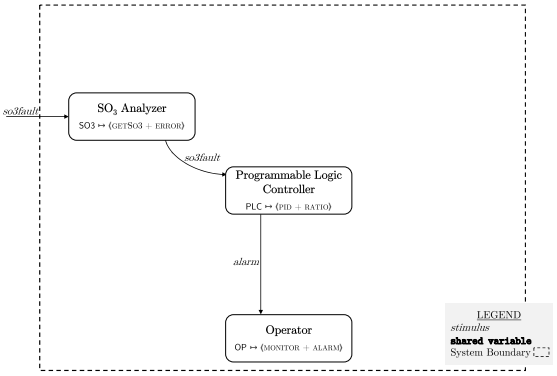
When we examine the communications that result from each stimulus received from outside the system boundary in isolation, we see the extent to which interference plays a role even further. In this context, interference now refers to interactions between the control flows that result from an external stimulus, rather than interactions between the pump-control and fault-management processes within the system. Figure 5.12 shows a breakdown of WDS4 into the communication that stems from each of the four external stimuli: *start*, *sapfault*, *so3fault*, and *repair*. Figure 5.12a is the same as Figure 5.11a, but Figures 5.12b, 5.12c, and 5.12d show Figure 5.11b broken down into three parts. None of the models in Figure 5.12 have any implicit interactions. What we find is that the implicit interactions in Figure 5.11b only appear when all



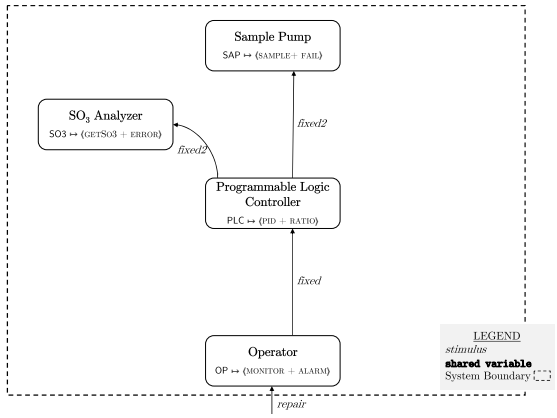
(a) Only communication that flows from the system receiving *start*



(b) Only communication that flows from the system receiving *sapfault*



(c) Only communication that flows from the system receiving *so3fault*



(d) Only communication that flows from the system receiving *repair*

Figure 5.12: Alternative designs of the WDS that each capture the communication flow from only one external stimuli

three of Figures 5.12b, 5.12c, and 5.12d are combined, just as how the majority of the implicit interactions in WDS4 only appear when combining the chemical feed pump control and fault management processes. The MCCS only has one external stimuli to initiate its' single process, so the issue of interference does not arise.

Additional WDS Design Alternatives

In the original WDS system design, all the stimuli for managing faults are centralized around PLC. We create another model transformation to fully centralize the system around PLC, by centralizing the pump-controlling communications around PLC as well over multiple steps With this, we can see if centralization has a similar impact to that of the MCCS transformation **T3**:

TVI Centralize the pump-controlling stimuli fully around PLC

- **TVIa** Centralize all pump-controlling stimuli around PLC. On receiving *start*, SAP still sends *eff*, but now PLC responds by sending *getdata*, which causes SO3 and SFM to perform the same work they would normally. Communication via shared environment is unchanged.
- **TVIb** Centralize all pump-controlling shared environments around PLC. Instead of SO3 and SFM directly referencing `effluent`, PLC now stores `effluent` in the shared environment `sapResult`. SO3 and SFM now reference `sapResult` where they previously referenced `effluent` in their concrete behaviors.

Four design alternatives are created by applying TVIa by itself, and another four by applying TVIa and TVIb together, to centralize the pump-controlling process in the original system design (WDS1), the WDS without oversights (WDS4), and the WDS with the additional FAULTMNGR agent (WDS12 and WDS13). As an example of the resulting system designs, the application of transformations TVIa and TVIb to WDS4 results in the system design **WDS17** shown in Figure 5.13. In this organization of

agents, the fault management process is unchanged, but now the communication flow for sampling and pump control is entirely coordinated by PLC. When we start with centralizing just the stimuli and leave the shared environments untouched, the number of implicit interactions in the system increases from 15 to 27, however, continuing to centralize the system through its shared environments as well decreases the number of implicit interactions in the system design again to 20.

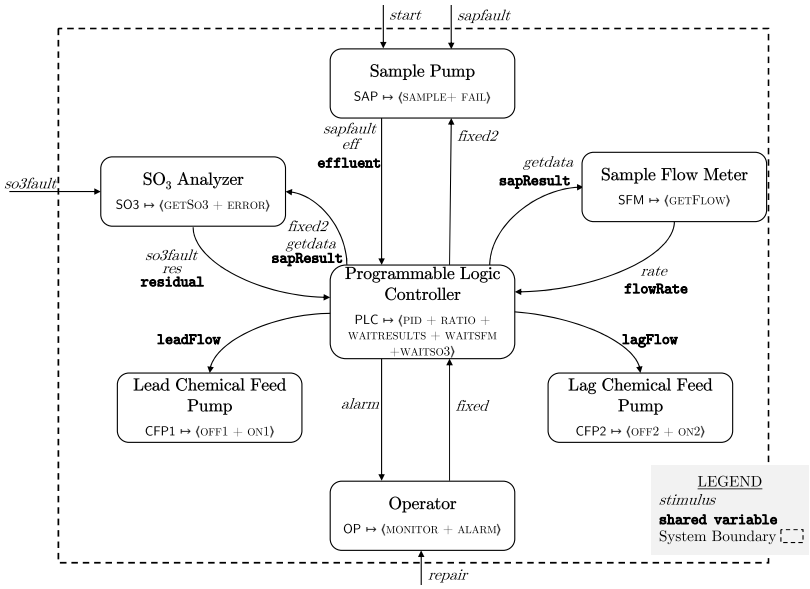
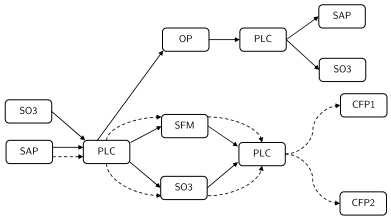


Figure 5.13: Alternative design **WDS17** with communication centralized around PLC after applying transformations **TI**, **TII**, **TVIa**, and **TVIb**

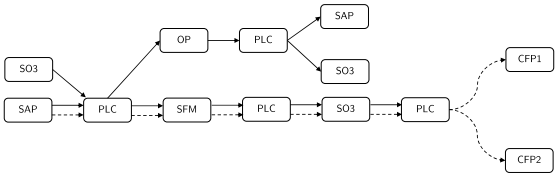
The final transformation we apply relates to concurrency in the system. The actions of SO3 and SFM are independent and have been designed to operate in parallel up until now, so we want to see how the prevalence of implicit interactions changes when we serialize their actions:

TVII Once SAP completes its actions, instead of both SO3 and SFM performing their behaviors in parallel, SFM will measure the flow rate and, only when SFM is finished, SO3 will perform its behaviors in the system.

We can combine this transformation with any alternate system design that has centralized all stimuli around PLC (TVIa), regardless of whether oversights are present (TII and TIII), the FAULTMNGR is included in the system (TV), or the shared environments are also centralized (TVIb). Applying transformation TVII to WDS17 in Figure 5.13 produces **WDS21**, which does not change the structure of the system design's communications, only their order. The difference is best shown in their intended interactions in Figure 5.14. Figure 5.14a shows the intended interactions for the system design when SFM and SO3 perform their behaviors in parallel, and figure 5.14b shows the intended interactions for the system design when SFM and SO3 perform their behaviors in series. Specifically in the case of WDS17 and WDS21, the number of implicit interactions increases from 20 in WDS17 when SFM and SO3 operate in parallel, to 24 in WDS21 when SFM and SO3 operate in series.



(a) Intended interactions of **WDS17**, where SO3 and SFM actions are performed in parallel



(b) Intended interactions of **WDS21**, where SO3 and SFM actions are performed in series

Figure 5.14: Alternative intended interactions of the WDS showing SO3 and SFM operating in parallel and in series

Implicit Interaction Dataset for the WDS Design Alternatives

Table 5.14 shows the prevalence of implicit interactions in each of the 29 total system design alternatives generated through combinations of the model transformations we identified. WDS1 is the original system design. WDS2 through WDS11 are generated from combining the different approaches to eliminating oversights. WDS12, WDS13, and WDS22 to WDS29 all include the additional FAULTMNGR agent. WDS14 through WDS29 all include one or both of the transformations to centralize the chemical feed pump-controlling communication around PLC. WDS18 to WDS21, WDS24, WDS25, WDS28, and WDS29 all serialize the actions of SO3 and SFM. Overall, WDS23 and WDS25, which include the FAULTMNGR and have oversights, have the highest prevalence of implicit interactions. WDS4 and WDS11, which only attempt to eliminate oversights and are the endpoints of applying the methods for identifying weak points in the WDS, have the lowest number of implicit interactions. For the WDS, we also include the number of oversights present in each alternate system design.

Table 5.14: Measurements of the prevalence of implicit interactions for each system design alternative for the WDS

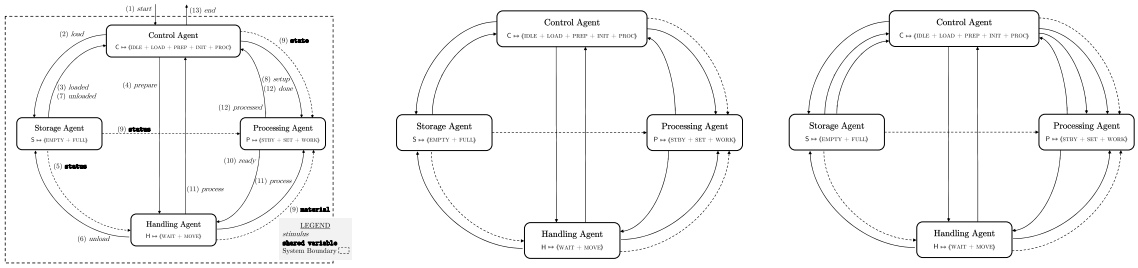
System Design Alternative	Transformations Applied	Number of Implicit Interactions	Percentage of Interactions that are Implicit	Number of Oversights
WDS1	-	74	52.48%	3
WDS2	**TI**	46	48.94%	2
WDS3	**TII**	29	36.25%	1
WDS4	**TI**, **TII**	15	28.30%	0
WDS5	**TIII**	37	33.64%	1
WDS6	**TIVa**	124	62.62%	4
WDS7	**TIVb**	97	57.74%	3
WDS8	**TI**, **TIVa**	63	56.25%	2
WDS9	**TI**, **TIVb**	68	57.63%	2
WDS10	**TI**, **TIII**, **TIVa**	19	23.17%	0
WDS11	**TI**, **TIII**, **TIVb**	15	19.48%	0
WDS12	**TV**	114	52.34%	4
WDS13	**TI**, **TII**, **TV**	23	31.08%	0
WDS14	**TVIa**	118	68.61%	4
WDS15	**TVIa**, **TVIb**	112	76.71%	4
WDS16	**TI**, **TII**, **TVIa**	27	41.54%	0
WDS17	**TI**, **TII**, **TVIa**, **TVIb**	20	37.04%	0
WDS18	**TVIa**, **TVII**	135	78.49%	4
WDS19	**TVIa**, **TVIb**, **TVII**	116	79.45%	4
WDS20	**TI**, **TII**, **TVIa**, **TVII**	32	49.23%	0
WDS21	**TI**, **TII**, **TVIa**, **TVIb**, **TVII**	24	44.44%	0
WDS22	**TV**, **TVIa**	223	63.90%	5
WDS23	**TV**, **TVIa**, **TVIb**	310	83.11%	5
WDS24	**TV**, **TVIa**, **TVII**	279	79.94%	5
WDS25	**TV**, **TVIa**, **TVIb**, **TVII**	323	86.60%	5
WDS26	**TI**, **TII**, **TV**, **TVIa**	48	39.34%	0
WDS27	**TI**, **TII**, **TV**, **TVIa**, **TVIb**	76	60.32%	0
WDS28	**TI**, **TII**, **TV**, **TVIa**, **TVII**	68	55.74%	0
WDS29	**TI**, **TII**, **TV**, **TVIa**, **TVIb**, **TVII**	82	65.08%	0

5.4 Graph-Based Analysis

The purpose of performing a graph-based analysis on the alternative designs generated in Section 5.3 is to discover what relationships exist with the prevalence of implicit interactions discussed in Section 6.2. For each of the 10 design alternatives for the MCCS and the 29 design alternatives for the WDS, we first build an *interaction model* of the system design to which graph-based measurements can be applied. Then, after identifying measurements to take based on system properties we are interested in examining in Section 5.4.2, we record the measurements for the MCCS and WDS in Section 5.4.3.

5.4.1 Interaction Models

The interaction model of a system design is a directed multi-graph, where the set of agents $\mathcal{A}$ is the set of nodes and there are two classes of edges: one for communication via stimuli and a second for communication via shared environment. Between any pair of nodes, there is one edge for each stimulus communicated and one edge for each shared variable defined by the source agent and referenced by the sink agent. This is not the same as using the set $\mathcal{P}_{direct}$ as the edges in the graph. While the graph $(\mathcal{A}, \mathcal{P}_{direct})$ is still a directed multi-graph, it contains at most one edge from each class between any two nodes. Given the interaction model of a system, we can remove duplicate edges from each edge class between nodes to obtain the graph $(\mathcal{A}, \mathcal{P}_{direct})$, which is a *communication graph* [PP05] of the system design. While the interaction



(a) Collaboration diagram (b) Communication graph (c) Interaction model

Figure 5.15: The difference between communication graphs, collaboration diagrams, and interactions graphs for **MCCS1**

model shows what information is communicated between agents, the communication graph only shows the communication channels that exist in the system.

Figure 5.15 shows the difference between collaboration diagrams, communication graphs, and the interaction model. In the original MCCS design shown in the collaboration diagram in Figure 5.15a, the difference lies in the direct interactions $\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$ and $\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$. In the communication graph in Figure 5.15b, this is treated as one solid edge from S to C and one solid edge from C to P, whereas the interaction model shown in Figure 5.15c contains two solid edges from S to C and two solid edges from C to P because the two labels on the edges $\mathsf{S} \rightarrow_{\mathcal{S}} \mathsf{C}$ and $\mathsf{C} \rightarrow_{\mathcal{S}} \mathsf{P}$ in the collaboration diagram represent two different stimuli. Besides a collaboration diagram, we can also build the interaction model, and therefore the communication graph, from a sequence diagram or through the formal model itself by using the set of agents $\mathcal{A}$ as nodes, determining edges for communication via stimuli from the stimulus-response specification of each agent, and determining edges for communication via shared environment from the concrete behavior specifications of each agent.

5.4.2 Measurable System Properties of Interest

Throughout our experiments, we take measurements of system properties for two purposes. First, we want to determine which system properties are related to, and possibly the root cause of, the prevalence of implicit interactions in a system design. Second, we want to obtain indicators of the prevalence of implicit interactions, such that if a measurement increases or decreases from one design iteration to the next, we can see that the prevalence of implicit interactions reliably increases or decreases as well.

To take a measurement of a system property, we adopt an approach similar to the method proposed by Cavano and McCall [CM78] for measuring software quality. We determine the property of the system in which we are interested, select an attribute of the system that would characterize this system property, and then take a quantitative measurement of that system attribute. The attribute must be measured in a consistent way, and a relationship between the system property and measurement must be clearly explained. We explain each of the system properties we are interested in the above style, discuss why we expect them to relate to the prevalence of implicit interactions in a system, and what type of relationship we expect to see.

System-Level Complexity

The idea that complexity relates to a higher prevalence of implicit interactions has been expressed in terms system size [JV17b], interconnectedness [JV17a], and hidden coupling [JV17a, Jas20a]. Therefore, we explore multiple different measurements that

represent the concept of system complexity to determine how complexity is relevant to the prevalence of implicit interactions.

We expect larger systems to have a higher prevalence of implicit interactions because if a system is composed of more parts, more considerations need to be made to avoid having interactions go unnoticed, and there are more opportunities for things to go wrong. We can measure complexity as related to size in two ways. First, we can use the cyclomatic complexity [McC76] of the communication graph:

$$CC = e - n + 2p \tag{5.4}$$

where e is the number of edges, n is the number of nodes, and p is the number of connected components. In the communication graph, e is the number of direct interactions in $\mathcal{P}_{direct}$ and n is the number of agents in $\mathcal{A}$. When a directed graph is connected (i.e., replacing all of its directed edges with undirected edges produces a path between each pair of nodes), there is one component, so p is 1. This is the case in all our design alternatives. Therefore, we can simplify the equation for our context to:

$$CC = |\mathcal{P}_{direct}| - |\mathcal{A}| + 2 \tag{5.5}$$

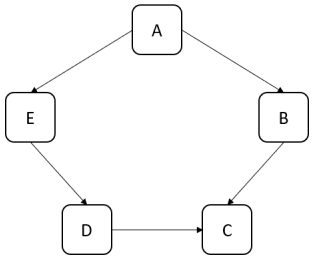
We can also measure complexity from the C^2KA representation of the system. The "size" of a system in this view would relate to how large the next-stimulus and next-behavior mappings are, which we can calculate by multiplying the total number of behaviors in a system by the number of stimuli in the system. The size of the C^2KA

specification is calculated as:

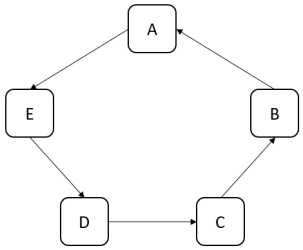
$$|\text{Spec}| = |K \backslash \{0, 1\}| \times |S \backslash \{\mathfrak{n}, \mathfrak{d}\}| \tag{5.6}$$

We can multiply the size of each set to get the size of the specification because the next-behavior and next-stimulus mappings are total functions. Every combination of stimulus and behavior in either mapping has an output stimulus or behavior, even if they are simply the identity or annihilating elements $\{0, 1\}$ from K or $\{\mathfrak{n}, \mathfrak{d}\}$ from S. We exclude these identity and annihilating elements from both sets when measuring the size of the C^{2}KA representation of the system because they will always be present in every system design represented with C^{2}KA. Their exclusion also allows for the measurement to be clearer from how we represent the next-behavior and next-stimulus mappings. For example, one can reach the same result as Equation 5.6 for the original MCCS design by counting the number of cells in the tabular representations of either mapping in Section 4.4. The cyclomatic complexity of a system design conveys the structural size of the system, as it measures system size in terms of how many components and communication channels a system is composed of, whereas the size of the C^{2}KA specification is more related to a behavioral view of complexity as it is concerned with what states and messages exist in the system.

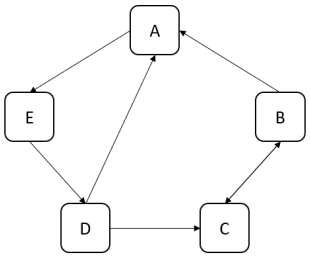
Finally, as implicit interactions are defined as paths and therefore do not repeat any agents in their communication sequence, we want to see how cycles in the communication graph of a system design relates to the prevalence of implicit interactions. We measure cycles in terms of the number that exist in the graph, the size of largest cycle, and the median length of cycles that exist in the graph. Agents that take part



(a) Communication graph with no cycles



(b) Communication graph with one large cycle ($\mathsf{A} \rightarrow \mathsf{E} \rightarrow \mathsf{D} \rightarrow \mathsf{C} \rightarrow \mathsf{B}$)



(c) Communication graph with one large cycle ($\mathsf{A} \rightarrow \mathsf{E} \rightarrow \mathsf{D} \rightarrow \mathsf{C} \rightarrow \mathsf{B}$) and two smaller cycles ($\mathsf{A} \rightarrow \mathsf{E} \rightarrow \mathsf{D}$ and $\mathsf{C} \rightarrow \mathsf{B}$)

Figure 5.16: Example systems to differing number and size of cycles

in cycles, as shown in Figure 5.16b and Figure 5.16c, have the potential to influence every other agent in that cycle, whereas in Figure 5.16a agents E and D have no communication paths that lead to B or A. We expect a system with a greater prevalence of implicit interactions to have more cycles and tend towards having cycles of a larger size.

Agent-Level Complexity

When considering complexity from a coupling perspective, we look at each agent in a system design, rather than a system design as a whole, to see how it relates to the prevalence of implicit interactions. We look at the coupling of an agent as how many ways it can communicate with other agents, and therefore we can count coupling for any given agent with its *degree centrality* in the interaction model of the system design. Degree centrality measures the total number of incoming and outgoing edges for a given node in the graph [BJG07]. For a directed graph, this can be broken

Table 5.15: Degree centrality values for each agent in the MCCS

Agent	Degree Centrality	In-Degree	Out-Degree
C	9	5	4
S	6	2	4
H	7	3	4
P	8	6	2

down into *in-degree*, which counts just the number of incoming edges for a node, and *out-degree*, which counts just just the number of outgoing edges for a node. In our context, in-degree is the sum of the number of incoming stimuli an agent receives and the number of shared environments it references, out-degree is the sum of the number of stimuli an agent sends and the number of shared environments it defines, and the degree centrality is the total number of direct communications via stimuli or shared environment in which the agent takes part. We can easily compute these centrality values for the MCCS by counting the edges in Figure 4.1, as shown in Table 5.15.

Since implicit interactions are described as hidden linkages between components [JV17b], we also consider eigenvector centrality and PageRank to measure global influence of an agent. The eigenvector centrality of a node more accurately captures the global importance of a node, where a node's centrality is assigned based on the centrality of neighboring nodes [Bon87]. These values can be computed using the following equation:

$$\lambda e = Re \tag{5.7}$$

where R is the adjacency matrix representing an undirected version of the interaction model with eigenvalue λ, and e is the eigenvector centrality values. The i-th value in e corresponds to the centrality value for the node represented by row i in R. For directed

Table 5.16: Eigenvector centrality values for each agent in the MCCS

Agent	Eigenvector Centrality	PageRank
C	0.5	0.2869
S	0.5	0.1399
H	0.5	0.2524
P	0.5	0.3208

graphs PageRank can be used instead, which takes into account the in-degrees of nodes instead of all edges [NH19]. Centrality methods give a set of values rather than one number so we cannot as easily compare them across different variations of a system design, but we can compare the centrality measures for each agent with how many implicit interactions each agent is present in to see if a higher centrality by any method relates to that agent occurring more frequently in the system's implicit interactions. As an example, the eigenvector and PageRank centrality values for the MCCS are given in table 5.16. We compute these values using the NetworkX package for Python, which uses an iterative solver [Net], so some small variation may exist between the values in Table 5.16 and those calculated with Equation 5.7.

Concurrency

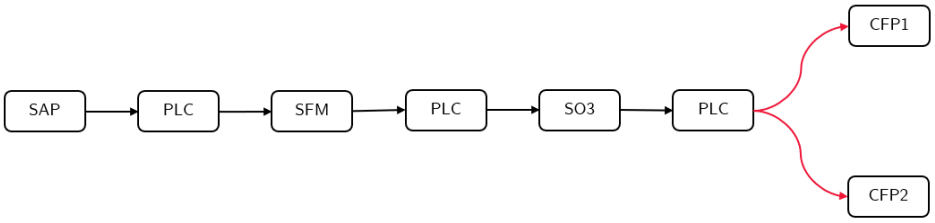
Implicit interactions represent possible out-of-order sequences of communication through a system, which are expected to be more prevalent in concurrent systems that have to synchronize activities in order to enforce the correct sequence of communication. The event trace that represents the intended interactions of a system (e.g., Figure 4.4) forks whenever the control flow branches with concurrent behaviors, which translates to more intended interactions in $\mathcal{P}_{intended}$. A greater amount of communication via

shared environments may be required when there is more synchronization required between agents, which also increases the number of intended interactions. Therefore, we can measure concurrency in a system by the amount of intended interactions in a system, and we expect to see a higher prevalence of intended interactions correlate with a higher number of intended interactions.

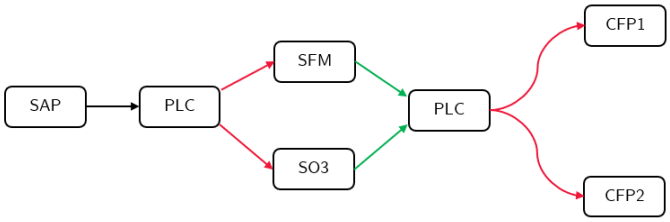
Figure 5.17 shows an example of how concurrency changes the number of intended interactions. To make this example more simple, only the control flow is shown rather than show both direct interactions via stimuli and via shared environment. Figure 5.17a shows the control flow in the intended interactions for the sampling and pump control process of the WDS and Figure 5.17b shows that same process with transformation **TVII** applied, serializing the actions of SFM and SO3. The concurrent actions of SFM and SO3 in Figure 5.17b doubles the number of walks through the graph shown compared to Figure 5.17a where each action is done in series.

Centralization

It is expected that a higher amount of centralization in a system will correlate with a higher prevalence of implicit interactions. This is because the agents around the centralized agent within a system may be able to influence others through the highly connected central agent. We can measure the centralization of a directed graph by computing the degree centrality of each node in the interaction model and finding the variance in these values [Sni81]. A smaller value indicates that more agents have a degree closer to the mean degree centrality within a system, so agents are similarly centralized, whereas a larger value means that a system has agents with



(a) With only one fork in the control flow (red), there are two intended interactions in this graph

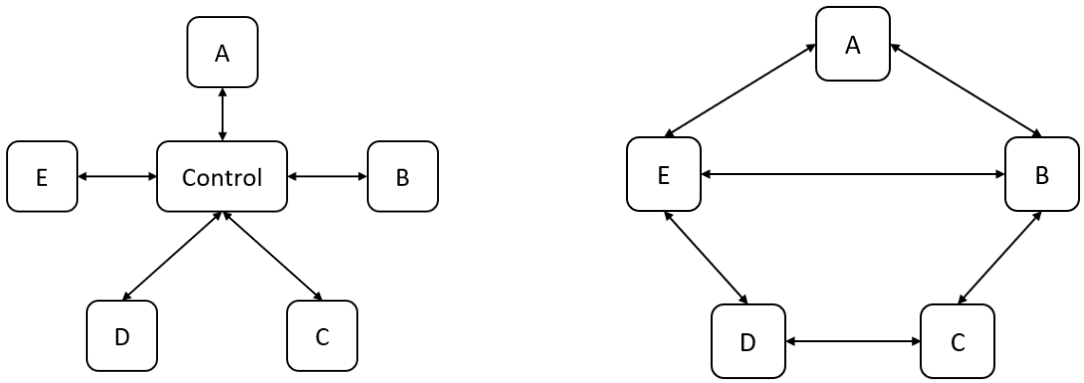


(b) With two forks in the control flow (red) and one join (green), this graph shows four intended interactions

Figure 5.17: Example systems to show how the number of intended interactions measures concurrency

degree centrality values farther from the mean degree centrality value, which occurs in such cases as when there is one highly centralized agent and many other agents with lower degree centrality values.

For example, in Figure 5.18, two versions of a hypothetical system are shown: one centralized around a control agent, and the other distributed. In the centralized system in Figure 5.18a, the variance of degree centrality values is 8.89, because the central control agent has a much higher degree centrality than all other agents, whereas the variance of degree centrality values in the distributed system shown in Figure 5.18b is only 0.96 as all agents have a similar degree centrality.



(a) Agents around a central control agent result in a high degree centrality variance of 8.89

(b) Agents with distributed control result in a much lower degree centrality variance of 0.96

Figure 5.18: Example centralized and distributed system to show how degree centrality variance measures centralization

We expect to see a larger variance in the degree centrality values of a system's detailed communication graph to correlate positively with the prevalence of implicit interactions in system designs.

5.4.3 Graph Measurement Datasets for MCCS and WDS System Design Alternatives

The measurements for each system design alternative of the MCCS are given in Table 5.17. Appendix A.1 contains the centrality measurements for each in each MCCS design alternative. The measurements for each system design alternative of the WDS are given in Table 5.18. Appendix A.2 contains the centrality measurements for each in each WDS design alternative. In both tables, in addition to the measurements described above, we also include the values used to calculate cyclomatic complexity

(i.e., the number of agents and number of direct interactions) and to calculate the size of the C^2KA specification (i.e., the number of stimuli and number of behaviors).

The two tables help to show some of the differences between the two systems, namely how larger measures of complexity for the WDS show the difference in size between the WDS and the MCCS. For the MCCS, most of the design alternatives are based solely on identifying weak points and we can see that the original design has both the highest cyclomatic complexity and most cycles of all the MCCS design alternatives. However, this is not true for the original WDS design, in which most design alternatives are not based solely on identifying weak points. We can see for both designs how the centralization measurement coincides with the design alternatives. MCCS2 to MCCS6, MCCS9, and MCCS10 all centralize more communication around the control agent, and so the variance in agent degree centrality is larger than MCCS1, whereas MCCS6 and MCCS7 specifically look to distribute system control so the variance in agent degree centrality is smaller than MCCS1. Many of the WDS design alternatives include centralizing more communication around PLC, which show a higher variance in agent degree centrality than the original system design.

Table 5.17: Graph-based measurements for each system design alternative for the MCCS

System Design Alternative	Number of Agents	Number of Direct Interactions	Cyclomatic Complexity	Number of Stimuli	Number of Behaviors	Size of C^2KA specification	Number of Cycles	Size of Largest cycle	Median Cycle Size	Number of Intended Interactions	Variance of Degree Centrality
MCCS1	4	13	11	12	12	144	15	4	3	16	1.6667
MCCS2	4	11	9	12	12	144	8	4	2.5	6	4.3333
MCCS3	4	12	10	12	10	120	8	4	2.5	12	8.3333
MCCS4	4	11	9	12	10	120	8	4	2.5	8	11.6667
MCCS5	4	8	6	16	12	192	3	2	2	8	40.6667
MCCS6	4	6	4	16	12	192	3	2	2	1	22.6667
MCCS7	3	8	7	7	6	42	3	3	2	6	0.3333
MCCS8	3	6	5	7	6	42	3	3	2	2	1
MCCS9	4	8	6	15	13	195	3	2	2	16	40.6667
MCCS10	4	6	4	15	13	195	3	2	2	2	22.6667

Table 5.18: Graph-based measurements for each system design alternative for the WDS

System Design Alternative	Number of Agents	Number of Direct Interactions	Cyclomatic Complexity	Number of Stimuli	Number of Behaviors	Size of C²KA specification	Number of Cycles	Size of Largest cycle	Median Cycle Size	Number of Intended Interactions	Variance of Degree Centrality
WDS1	7	18	13	12	13	156	14	5	3	36	14.4761
WDS2	7	17	12	13	13	169	9	4	3	40	10.9524
WDS3	7	16	11	13	13	169	8	4	2.5	40	13.6190
WDS4	7	15	10	14	13	182	5	3	2	40	10.6667
WDS5	7	16	11	12	13	156	8	5	3.5	45	9.6190
WDS6	7	20	15	12	13	156	25	5	3	38	16.5714
WDS7	7	19	14	11	13	143	18	5	3	24	15.2857
WDS8	7	18	13	13	13	169	12	4	3	38	11.8095
WDS9	7	18	13	12	13	156	13	5	3	24	10.8095
WDS10	7	16	11	13	13	169	6	4	3	48	7.5714
WDS11	7	16	11	12	13	156	3	3	2	30	6.9048
WDS12	8	21	15	12	15	180	10	4	3	31	10.5000
WDS13	8	17	11	14	15	210	4	3	2	31	6.8571
WDS14	7	18	13	13	16	208	15	5	3	28	25.3333
WDS15	7	19	14	13	16	208	11	4	3	54	35.2381
WDS16	7	14	9	15	16	240	6	3	2	28	23.8095
WDS17	7	15	10	15	16	240	4	2	2	54	35.4762
WDS18	7	18	13	14	15	210	15	5	3	32	25.3333
WDS19	7	19	14	14	15	210	11	4	3	102	35.2381
WDS20	7	14	9	16	15	240	6	3	2	32	23.8095
WDS21	7	15	10	16	15	240	4	2	2	102	31.8095
WDS22	8	23	17	13	18	234	22	6	3	31	15.4286
WDS23	8	24	18	13	18	234	19	5	3	57	21.3571
WDS24	8	23	17	14	17	238	22	6	3	50	15.4286
WDS25	8	24	18	14	17	238	19	5	3	149	21.3571
WDS26	8	18	12	15	18	270	9	5	2	31	12.5000
WDS27	8	19	13	15	18	270	7	4	2	57	19.1429
WDS28	8	18	12	16	17	272	9	5	2	50	12.5000
WDS29	8	19	13	16	17	272	7	4	2	149	19.1429

5.5 Conclusions

The purpose of this chapter was to generate all the data required to identify relationships between the prevalence of implicit interactions in a system design and other measurements taken from a semi-formal, graph-based analysis. We presented each activity required to generate this data, which consists of both formal and graph-based methods to model alternate system designs and measure properties. Specifically, the main takeaways from this chapter are:

1. We generate 10 design alternatives of the MCCS through combinations of the model transformations **T1**, **T2**, **T3**, **T4**, **T5**, and **T6**.

2. We generate 29 design alternatives of the WDS through combinations of the model transformations **TI**, **TII**, **TIII**, **TIV**, **TV**, **TVI** and **TVII**.

3. For each design alternative, we measure the prevalence of implicit interactions, given in Table 5.10 for the MCCS and Table 5.14 for the WDS.

4. For each design alternative, we measure graph-based measurements, given in Table 5.17 and Appendix A.1 for the MCCS, and Table 5.18 and Appendix A.2 for the WDS.

5. For both the MCCS and WDS, the design alternatives with the fewest implicit interactions are the final designs produced from considering only the formal analysis for identifying weak points.

In the next chapter, we compare Table 5.10 with Table 5.17, and Table 5.14 with Table 5.18 to determine which graph-based measurements relate to the implicit interaction measurements for the MCCS and WDS, and ultimately determine which system properties and design principles impact the prevalence of implicit interactions in a system design.

Chapter 6

Data Analysis Activities

In this chapter, we detail the data analysis activities presented in Chapter 3. Section 6.1 shows how we compare design alternatives to each other to identify specific changes that relate to differences in system properties to improve our understanding of implicit interactions, and Section 6.2 shows how we use the data generated by the formal and graph-based analyses to identify relationships between the two methods to answer **RQ2**. In each section, we describe the approach taken in the activity, and show its application on our two illustrative examples: the manufacturing cell control system and wastewater dechlorination system.

6.1 Compare Designs

With several design alternatives of each system, we now revisit those changes made through a more general design-focused lens to examine side effects that occur through

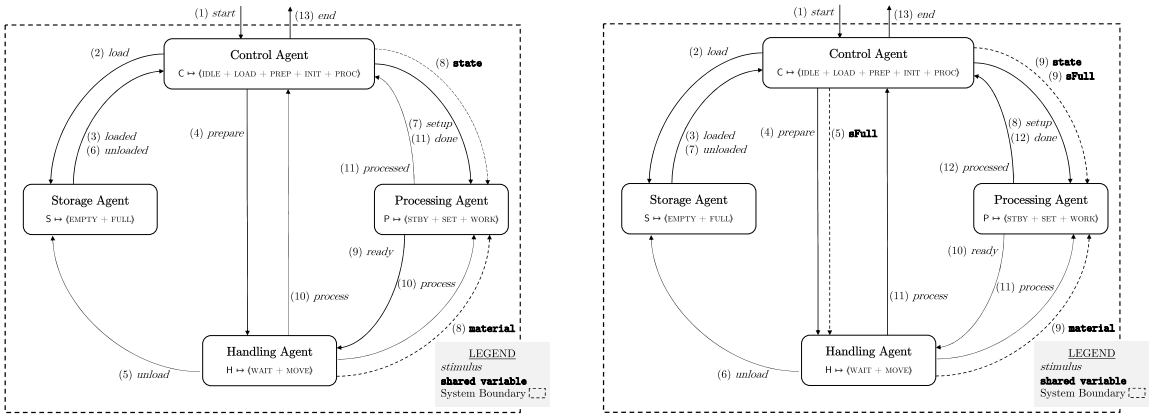
the application of model transformations. We compare groups of design alternatives to start working towards our ultimate goal of identifying design principles.

6.1.1 Comparing Design Alternatives for the MCCS

In the MCCS, there are relatively few design alternatives compared to the WDS, and a small enough number that we can easily compare pairs of alternate designs side-by-side to discuss how system properties vary.

Comparing Initial Design Changes to Eliminate Weak Points

The first set of changes we propose to the original MCCS design achieve the same goal of eliminating a specific weak point using alternate methods. While MCCS2, shown in Figure 6.1a, eliminates references to a shared environment, MCCS3, shown in Figure 6.1b, moves the place in which the shared environment is defined. Compared to the original system design, as shown in Table 5.10, both MCCS2 and MCCS3 have a reduced prevalence of implicit interactions, with MCCS2 reducing the prevalence of implicit interactions to a greater extent. This is understandable, as while both alternate designs eliminate weak points, MCCS2 is simpler in that it now has fewer communication channels, whereas MCCS3 has only restructured the communication channels to centralize more of the design around C. In both alternate designs, S and P are decoupled as there is no longer any direct communication between them. This may make more sense for the software design as in the physical process, the physical components controlled by H are what actually facilitate a "shared environment" by



(a) Collaboration diagram of **MCCS2** (b) Collaboration diagram of **MCCS3**

Figure 6.1: Alternative designs to eliminate weak points $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{P}$ and $\mathsf{S} \rightarrow_{\mathcal{E}} \mathsf{H}$ in the MCCS

moving the material that the components controlled by S store and the components controlled by P process. In both cases, S and H are also decoupled as there is now only one communication path between them.

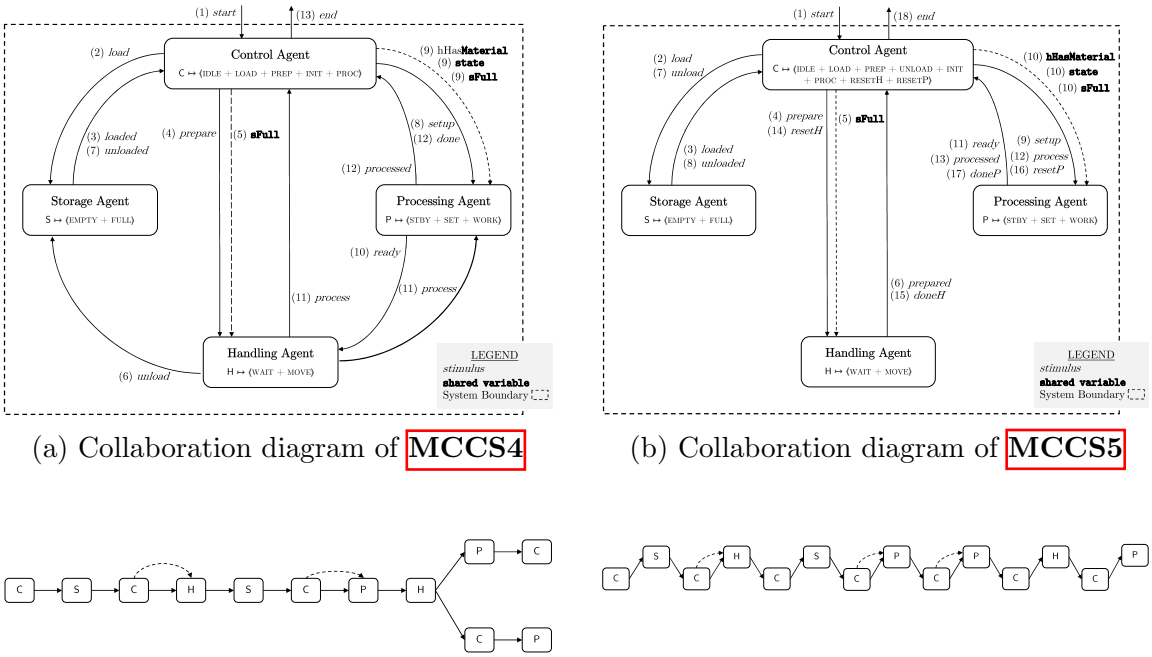
The additional communication channels created by `status` served as a way to improve the reliability of the design. In the original design, H and P would not act on only stimuli received from C, but also required a condition to be true about S. By eliminating the shared environment in MCCS2 completely, we have lost the extra reliability that condition would provide to the system, whereas in MCCS3, the additional conditions still exist but the shared environment resides in the same agent that issues the stimuli to trigger H or P to check this additional behavior. If C were compromised in MCCS1 then S would also need to be in the correct state for H or P to perform actions based on stimuli sent from C, whereas in MCCS2 and MCCS3, if C is compromised by an attacker it can more easily influence H and P by just sending the

appropriate stimuli in MCCS2, or by sending stimuli and redefining its own shared environments in MCCS3.

The Impact of Continued Centralization on the MCCS

MCCS4 and MCCS5 build on MCCS3 by continuing to centralize the system. MCCS4, shown in Figure 6.2a, and MCCS5, shown in Figure 6.2b, each reduce the number of implicit interactions in the system further, from 19 in MCCS3 to 17 in MCCS4 to only 3 in MCCS5. The percentage of total interactions that are implicit is less consistent, increasing slightly from MCCS3 to MCCS4 before dropping sharply in MCCS5. The initial increase in prevalence of implicit interactions is okay when we consider that MCCS4 is primarily a stepping-stone from MCCS3 to MCCS5. In MCCS4, we centralize only one form of communication and this inconsistency in the design might lead to an increased prevalence of implicit interactions.

The continued centralization around C decouples the other agents from each other while increasing the coupling of C, and is more in line with the expected responsibility of a component labelled as a control agent. While in MCCS1 to MCCS4, most agents coordinated operations with each other, in MCCS5 the control agent has full control of system operations and the other agents focus only on the actions related to their physical components. The more centralized, cohesive design extends not only to the control agent but others as well, because now the storage, handling, and processing agents have only the responsibility given to the based on their names, instead of also coordinating the actions of other agents.



(a) Collaboration diagram of **MCCS4**

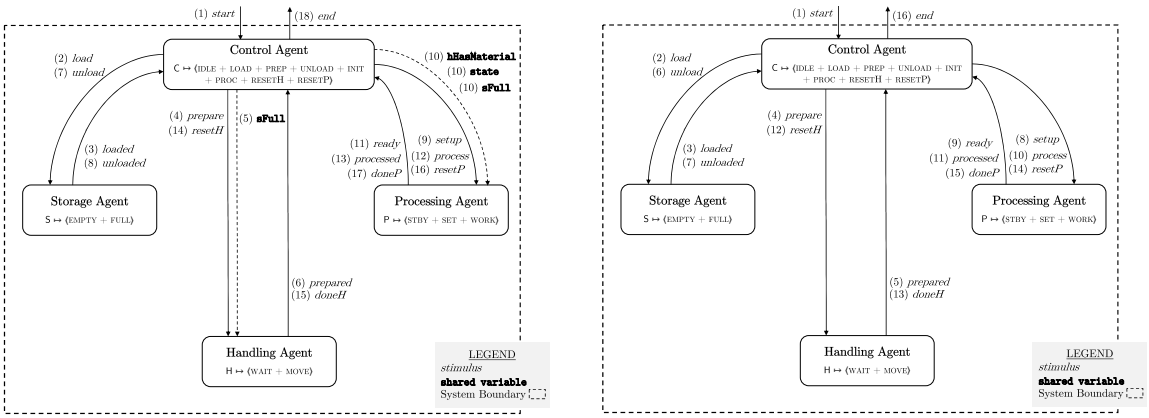
(b) Collaboration diagram of **MCCS5**

(c) Intended interactions of **MCCS4**

(d) Intended interactions of **MCCS5**

Figure 6.2: Alternative designs to gradually centralize the MCCS around C

We can see how the intended interactions compare through Figure 6.2c and Figure 6.2d, or through the numbering on the edges of the collaboration diagrams in Figure 6.2a and Figure 6.2b. Centralizing communication around C, while reducing the prevalence of implicit interactions, removes the concurrency that occurs in MCCS4 and previous alternate designs and also increases the overall length of the intended interactions as everything must now pass through C. If this system has strict performance requirements, this may not be desirable as there is more communication and thus more delay in each cycle of system operations. Centralizing everything around C also makes this agent a much more attractive target since if this agent is compromised it can potentially invoke any action the system as a whole is capable of performing.

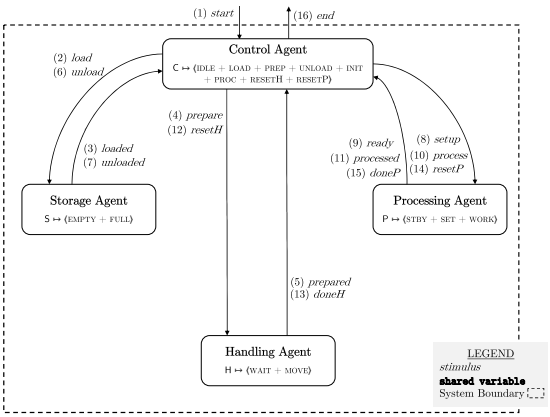


(a) Collaboration diagram of **MCCS5** (b) Collaboration diagram of **MCCS6**

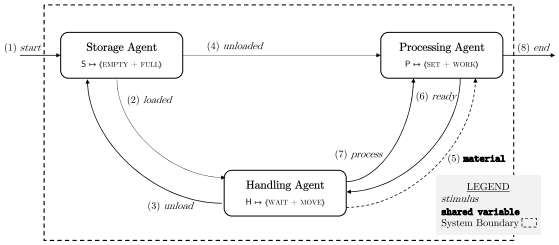
Figure 6.3: Alternative centralized MCCS designs with and without shared environments

The Impact of Eliminating Unnecessary Communication Channels

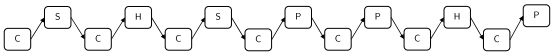
We remove the shared environments in MCCS5 to produce MCCS6, shown in Figure 6.3. Doing so eliminates two of the remaining three implicit interactions in MCCS5, and greatly simplifies communication in the system. Similar to the alternate designs in Figure 6.1, removing the shared environments can be seen as a reliability issue as removing the shared environments the H and P reference removes additional conditions they can check to ensure the system is in a correct state before performing any operations. Like we point out with MCCS3 however, the shared variables stored in C do not necessarily reflect the exact state of S at any time, so these shared variables may not be all that helpful and the design change can be viewed as simply eliminating unnecessary additional communications to reduce the attack surface of the system.



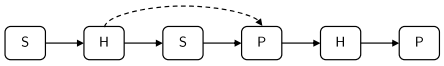
(a) Collaboration diagram of **MCCS6**



(b) Collaboration diagram of **MCCS8**



(c) Intended interactions of **MCCS6**



(d) Intended interactions of **MCCS8**

Figure 6.4: Alternative designs to centralize and distribute control in the MCCS

MCCS7 and MCCS8 are meant to take the original MCCS design in a more distributed direction. The direction of C simplifies the system in terms of the number of agents and direct interactions that exist as now S, H, and P all directly communicate instead of passing communication through C. Both greatly reduce the prevalence of implicit interactions in the system compared to original, with MCCS6 having a slightly high number of implicit interactions and MCCS8 having a higher percentage of total interactions that are implicit.

The simplified system can be seen in Figure 6.4a and Figure 6.4b in how much fewer communication occurs in Figure 6.4b. The intended interactions for MCCS6, shown in Figure 6.4c, and MCCS8, shown in Figure 6.4d, help to highlight the difference in communication overhead. Both involve the same order of non-control agents, but because MCCS8 has no control agent to route communications through, the intended

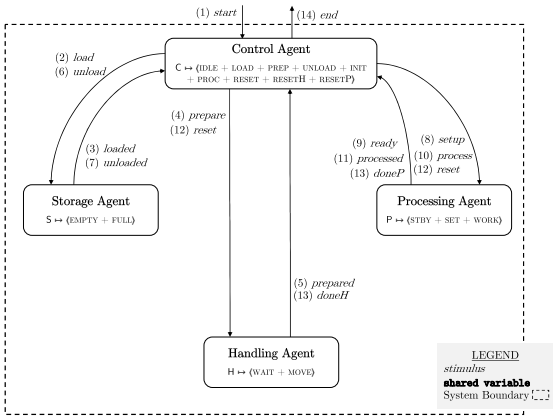
interactions are much shorter. Like when comparing MCCS4 and MCCS5, the shorter intended interactions and less communication overhead points towards improved performance in MCCS8 over MCCS1 and MCCS6. In MCCS6, C is by far the most appealing target for attackers to attempt to compromise whereas the distributed control in MCCS8 spreads out control between the three agents.

Comparing Concurrent and Sequential Control Flows in the MCCS

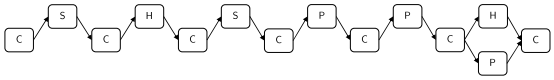
Finally, introducing concurrency into MCCS5 and MCCS6 to create MCCS9 and MCCS10, respectively, consistently increases the prevalence of implicit interactions in the system design. MCCS10 has a lower prevalence of implicit interactions than MCCS9, just like MCCS6 has a lower prevalence of implicit interactions than MCCS5. Resetting H and P in series in MCCS5 and MCCS6 results in an extra round trip of communications between C and non-control agents, resulting in potentially lower performance than MCCS9 and MCCS10, which can be seen by comparing the intended interactions of MCCS6 in Figure 6.4c against the intended interactions of MCCS10 in Figure 6.5b. Introducing concurrency also adds to the complexity of MCCS9 and MCCS10, as we require more complicated behaviors in C to handle H or P completing their concurrent actions in either order.

6.1.2 Comparing Design Alternatives for the WDS

In the WDS we have more design alternatives than in the case of the MCCS, but due to how we produce those design alternatives through different combinations of model



(a) Collaboration diagram of **MCCS10**



(b) Intended interactions of **MCCS10**

Figure 6.5: **MCCS6** with H and P reset in parallel at the end of the system operations (**MCCS10**)

transformations, we have many analogous design alternatives that differ by only one model transformation. This allows us to see how that specific model transformation impacts aspects of the system design in many different contexts.

Comparing Designs With and Without Oversights Present

Eliminating oversights from system designs has the greatest impact on the prevalence of implicit interactions for the model transformations we apply to the WDS. In every instance where we remove all oversights from a system design, shown in Table 6.1, the number of implicit interactions and prevalence of implicit interactions decrease drastically. When oversights are present, in Table 5.10 we see that alternate designs with more oversights always have more implicit interactions, but it is not always the case that designs with no oversights at all have fewer implicit interactions than other designs that have oversights. For example, WDS29 has 82 implicit interactions, the most of any design alternative without oversights. While eliminating all

Table 6.1: Change in the prevalence of implicit interactions when oversights are removed from the WDS

System Design With Oversights	Corresponding Design Without Oversights	Change in Number of Implicit Interactions	Change in Prevalence of Implicit Interactions
WDS1	**WDS4**	-59	-24.18%
WDS1	**WDS11**	-59	-33.00%
WDS12	**WDS13**	-91	-22.19%
WDS14	**WDS16**	-91	-27.07%
WDS15	**WDS17**	-92	-39.68%
WDS18	**WDS20**	-103	-29.26%
WDS19	**WDS21**	-92	-35.01%
WDS22	**WDS26**	-175	-24.55%
WDS23	**WDS27**	-234	-22.79%
WDS24	**WDS28**	-211	-24.21%
WDS25	**WDS29**	-241	-21.52%

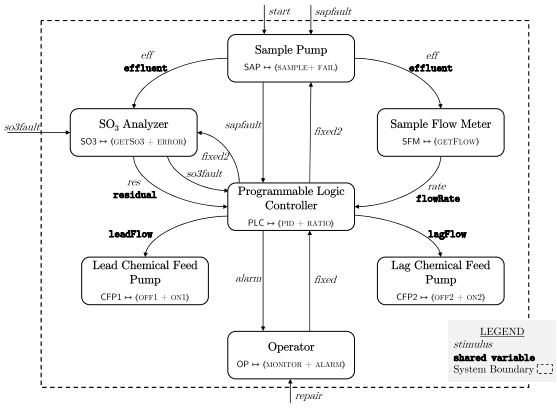
oversights produces a large difference in the implicit interactions measurements taken from the same system with oversights (WDS25), WDS3, which has the fewest implicit interactions of any design with oversights present, has only 29 implicit interactions. Oversights are not the only difference between WDS3 and WDS29 of course, and the higher number of implicit interactions in WDS29 might be explained through the inclusion of the FAULTMNGR, which results in a larger system.

Since oversights are defined as additional direct interactions that exist in a system but do not contribute to the system's objectives, their removal is an exercise in simplifying the system design by eliminating unnecessary potential communication like we do in the MCCS transforming MCCS5 to MCCS6. Removing communication between agents also reduces the coupling between those agents, as well as overall coupling in the system.

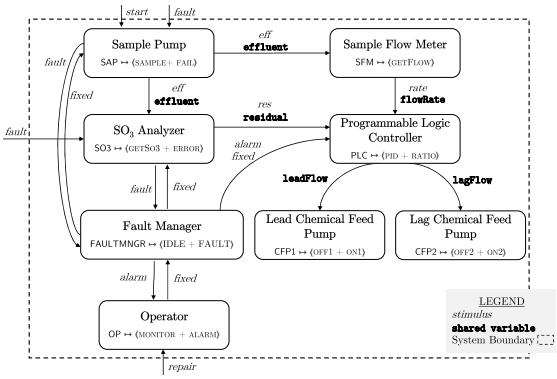
Comparing Designs With and Without the Fault Manager Agent

While the introduction of the eighth agent FAULTMNGR in WDS12 and WDS13 helps to separate the responsibilities of controlling pumps and managing fault-related stimuli that PLC otherwise must perform by itself, both WDS12 and WDS13 have a higher prevalence of implicit interactions than their seven-agent counterparts WDS1 and WDS4. It could be that the addition of an extra agent simply increases the attack surface of the system too much. An extra agent means we now have to consider how every other agent might implicitly influence, or be influenced by, that new agent, and can increase the overall system coupling despite improving cohesion within other agents. For example, the addition of FAULTMNGR to WDS4 to produce WDS29 actually eliminates all 15 implicit interactions that exist in WDS4, but introduces 23 implicit interactions that all pass through the new eighth agent, leading to an overall increase in the number of implicit interactions.

While a system design with FAULTMMGR, such as Figure 6.6b no longer sends fault-related stimuli from SAP and SO3 to PLC as it does in Figure 6.6a, SO3 can still directly interact with PLC to send the results of the sampling SO3 performs for PLC to manage pumps. FAULTMNGR also now needs to send stimuli to PLC to switch between PID and RATIO modes in Figure 6.6b on top of sending the *fixed* stimulus to each agent PLC needs to in in Figure 6.6a. This results in a design that overall has more coupling and is more complex. Unlike the MCCS where cohesion is improved by shifting responsibilities around existing agents, the changes in cohesion that FAULTMNGR provides are not sufficient to offset that additional complexity and overall system coupling that comes with the addition of a new agent.



(a) **WDS4** in which fault-related stimuli and managed by PLC



(b) **WDS12** in which fault-related stimuli and managed by FAULTMNGR

Figure 6.6: WDS design alternatives with and without FAULTMNGR

Table 6.2 shows the difference in implicit interaction measurements for otherwise equivalent system designs with and without FAULTMNGR. In all cases, both measurements increase when FAULTMNGR is added to the system design. The number of implicit interactions increases most when there are oversights present in the system, as is the case with WDS12, WDS22, WDS23, WDS24, and WDS25. This is because FAULTMNGR propagates a fixed message to all three of SAP, SO3, and PLC. When this propagation is done incorrectly to result in oversights, there are three oversights FAULTMNGR $\rightarrow_{\mathcal{S}}$ SAP, FAULTMNGR $\rightarrow_{\mathcal{S}}$ SO3, and FAULTMNGRPLC $\rightarrow_{\mathcal{S}}$ whereas when PLC handles the *fixed* propagation incorrectly it only results in oversights PLC $\rightarrow_{\mathcal{S}}$ SAP and PLC $\rightarrow_{\mathcal{S}}$ SO3.

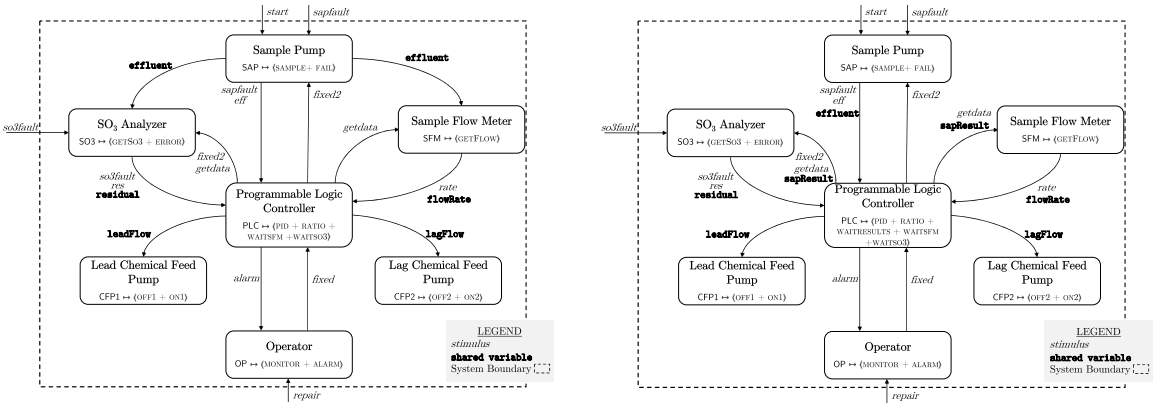
Table 6.2: Change in the prevalence of implicit interactions when FAULTMNGR is added to the WDS

System Design	Corresponding Design With FAULTMNGR	Change in Number of Implicit Interactions	Change in Prevalence of Implicit Interactions
WDS1	**WDS12**	+40	+0.79%
WDS4	**WDS13**	+8	+2.78%
WDS14	**WDS22**	+105	+4.71%
WDS15	**WDS23**	+198	+6.40%
WDS16	**WDS26**	+21	+2.19%
WDS17	**WDS27**	+56	+23.28%
WDS18	**WDS24**	+144	+1.45%
WDS19	**WDS25**	+207	+7.14%
WDS20	**WDS28**	+36	+6.51%
WDS21	**WDS29**	+58	+20.64%

The Impact of Continued Centralization in the WDS

The purpose of transformations TVIa and TVIb were to centralize the system in a similar manner to what was done in the MCCS to see if this also produces alternative designs with a lower prevalence of implicit interactions. From WDS4, when we first centralize the communication via stimuli in the system around PLC to produce Figure 6.7a, we see the same effect as in the MCCS where the prevalence of implicit interactions is higher with only one type of communication centralized. However, unlike the MCCS, when we centralize all communication around PLC to produce Figure 6.7b, the prevalence of implicit interactions is higher in the WDS when communication is not centralized. This tells us that simply the act of centralization is not what caused the prevalence of implicit interactions to drop in the MCCS. In the MCCS, the weak points direct us to centralizing the system, so the prevalence of

implicit interactions drops, whereas weak points do not point us in this direction for the WDS.



(a) **WDS16** which centralizes communication by stimuli around PLC

(b) **WDS17** which centralizes all communication around PLC

Figure 6.7: WDS alternate design to centralize communication around PLC

Centralization of the pump-controlling process consistently increases the implicit interaction measurements, as shown in Table 6.3. Each row in Table 6.3 shows one system design alternative, a second design alternative which is equivalent to the first with model transformations TVIa and TVIb applied, and the resulting change in implicit interactions. Like Table 6.2, the increase in implicit interactions is more pronounced when oversights are present in the systems being centralized.

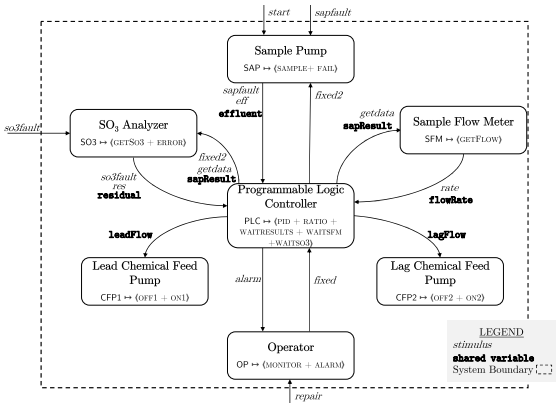
Table 6.3: Change in the prevalence of implicit interactions when centralizing the pump-controlling communications in the WDS

System Design	Corresponding Centralized Design	Change in Number of Implicit Interactions	Change in Prevalence of Implicit Interactions
WDS1	**WDS15**	+38	+24.23%
WDS4	**WDS17**	+5	+8.74%
WDS12	**WDS23**	+196	+29.84%
WDS13	**WDS27**	+45	+24.66%

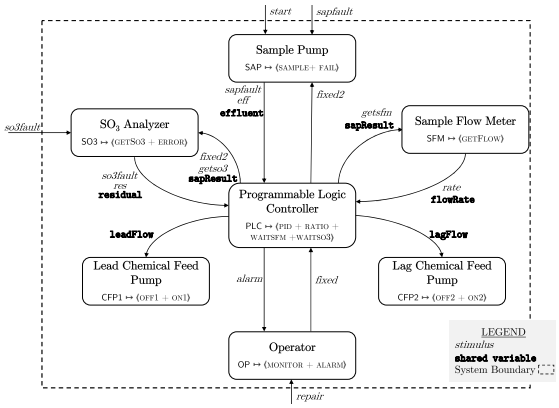
Looking more closely at the intended interactions of alternate designs for the MCCS and WDS also helps to explain why centralizing the design helps reduce the implicit interactions in the MCCS but not the WDS. With designs that have no oversights and are completely centralized around one agent, any implicit interactions that exist can only start in one non-central agent (subsequently referred to as a *leaf*[1] agent), propagate through the central control agent, and end in another leaf agent. With the WDS, the intended interactions (e.g., Figure 6.8c) visit each of the leaf agents only once or twice and cover relatively few of every possible pair of leaf agents before and after the central PLC, leading to many implicit interactions. The intended interactions for the centralized MCCS (e.g., Figure 6.4c) visit each leaf agent multiple times and cover a relatively large number of the possible pairs of leaf agents in the system design, so there are few implicit interactions.

From a higher level view, what we see is that the centralization in the WDS is more tightly coupling a greater number of agents together through the central PLC, though the coupling is still indirect. For example, SO3 and SFM cannot directly communicate in either the original system design or centralized system design, but in the centralized design such as Figure 6.8a there are many combinations of different stimuli or shared environments that can form short communication paths between the two agents in either direction. More combinations of stimuli and shared environments also exist in Figure 6.8a to realize the paths from SAP to CFP1 or CFP2 when compared to the original system design.

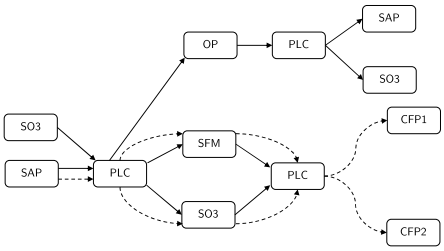
[1] The term *leaf* originates in graph theory, where a star graph of N nodes has one central node and $N-1$ leaf nodes [MFH14]. Centralized systems designs resemble a star graph, such as Figure 6.8a, where PLC is the central node and the other 6 agents are leaf nodes.



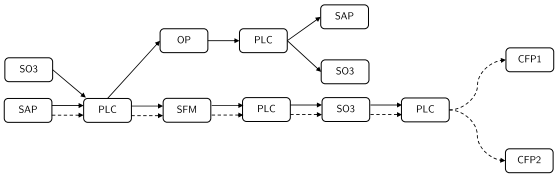
(a) Centralized WDS with SFM and SO3 actions in parallel (**WDS17**)



(b) Centralized WDS with SFM and SO3 actions in series (**WDS21**)



(c) Intended interactions of **WDS17**



(d) Intended interactions of **WDS21**

Figure 6.8: A comparison of alternate WDS designs with the actions of SFM and SO3 in parallel and in series

Comparing Concurrent and Sequential Control Flows in the WDS

We also create design alternatives where the actions of SFM and SO3 are either performed in series or in parallel so that we can compare the change in the prevalence of implicit interactions with a similar change in the MCCS. With or without oversights present in the system, performing the actions of SFM and SO3 in series, such as in Figure 6.8b and Figure 6.8d, rather than in parallel, such as in Figure 6.8a and Figure 6.8c, results in a higher prevalence of implicit interactions, which is opposite to the results we obtain from the portion of the MCCS we parallelize and serialize.

Table 6.4 shows the otherwise equivalent system designs with the actions of SFM and SO3 in parallel and in series, showing a consistent increase in implicit interactions whenever the actions of SFM and SO3 are serialized.

Table 6.4: Change in the prevalence of implicit interactions when applying **TVII** to the WDS

System Design	Corresponding Design with Serialized SFM/SO3	Change in Number of Implicit Interactions	Change in Prevalence of Implicit Interactions
WDS14	**WDS18**	+17	+9.88%
WDS15	**WDS19**	+4	+2.74%
WDS16	**WDS20**	+5	+7.69%
WDS17	**WDS21**	+4	+7.41%
WDS22	**WDS24**	+56	+16.05%
WDS23	**WDS25**	+13	+3.49%
WDS26	**WDS28**	+60	+16.39%
WDS27	**WDS29**	+6	+4.76%

Like with centralization, this tells us that reducing concurrency does not necessarily reduce implicit interactions like in the case of the MCCS. In the case of the WDS, this again comes down to the leaf agents in the centralized design. Both Figure 6.8a and Figure 6.8b have similar structures, but because of the alternating forks and joins in the intended interactions in Figure 6.8c, a lot more possible system paths are covered when compared to the intended interactions in Figure 6.8d. For example, the intended interactions in Figure 6.8d no longer includes paths SAP $\rightarrow$ PLC $\rightarrow$ SO3, SFM $\rightarrow$ PLC $\rightarrow$ CFP1, or SFM $\rightarrow$ PLC $\rightarrow$ CFP2 which all exist in Figure 6.8c.

The Impact of Multiple Overlapping Design Changes

Though many of the changes we make to the WDS worsen the number of implicit interactions, we can think in terms of how reverting those changes improves the

number of implicit interactions. For example, in Table 6.2 we see that adding the agent FAULTMNGR consistently increases the number of implicit interactions, but we can also say that removing FAULTMNGR from each design consistently reduces the number of implicit interactions. With this lens, we can look at how multiple model transformations, or the inverse of a presented transformation, improves implicit interactions over many steps, and whether multiple design changes compound to produce better, worse, or equivalent results to the individual design changes discussed up until now in previous subsections.

In Table 6.5 we start with WDS25, which has oversights present, contains the extra fault manager agent, has centralized the pump-controlling process fully around PLC, and has serialized the actions of SFM and SO3 in the pump-controlling process. Following WDS25 are a series of other design alternatives that change one of these four aspects of the design until ending with WDS4 which has no oversights, no FAULTMNGR, no additional centralization, and no serialization of actions. For each design, we show the total number of implicit interactions, along with how many implicit interactions that exist in the previous row's design alternative no longer exist and how many new implicit interactions the design change introduces.

It is important to note that we can make these changes in any order to move from WDS25 to WDS4. If transformations were made in a different order, the rows between WDS25 and WDS4 would include different design alternatives to include those with different combinations of features, but the conclusions are the same. In this context, we also refer to the implicit interactions that no longer exist from one design alternative to the next as "eliminated" and those that did not exist in the previous

design alternative as "new" despite not creating the design alternatives in the order shown in Table 6.5. What this actually means is that, for example, WDS23 has 238 implicit interactions not present in WDS12 and WDS12 has 42 implicit interactions not present in WDS23, but for the purposes here it is useful to use the words "eliminated" and "new" to talk about gradually progressing from WDS25 to WDS4.

Table 6.5: Change in the number of implicit interactions over many model transformations

System Design	Oversights Present?	FAULTMNGR Present?	Fully Centralized Pump-Control?	SFM and SO3 in series?	Implicit Interactions "Eliminated"	"New" Implicit Interactions	Total Implicit Interactions
WDS25	✓	✓	✓	✓	–	–	323
WDS23	✓	✓	✓	✗	21	8	310
WDS12	✓	✓	✗	✗	238	42	114
WDS1	✓	✗	✗	✗	95	55	74
WDS4	✗	✗	✗	✗	59	0	15

Table 6.5 helps to show how any design change can not only remove implicit interactions, but also introduce new implicit interactions. Because each change introduces new implicit interactions, the effect of making multiple changes is not equal to the sum of impacts from making each individual change. For example, though the difference in number of implicit interactions between WDS25 and WDS4 is 308, summing the column for "eliminating" implicit interactions totals to 413 because most changes also introduce new implicit interactions. However, subsequent rows eliminate more implicit interactions than they introduce, including those introduced by previous rows to yield a lower and lower number of implicit interactions as more design changes are made. Therefore, while not a simple linear relationship, considering multiple design changes to mitigate the existence of implicit interactions in a system design is useful since different design changes target different implicit interactions.

6.2 Identifying Relationships

We primarily determine whether or not a relationship exists through *Spearman's rank correlation* (also referred to as simply *Spearman correlation*), which describes how monotonically related two sets of values are [CF14]. As we cannot possibly try every possible design change for a system, we rely on critical value tables to determine whether the Spearman's rank correlation values we obtain are statistically significant with $\alpha = 0.05$. Critical value tables show the threshold value for a correlation at which one can be confident with a specified certainty that the relationship between two sets of values of a given size in fact exists, and is not due to random variation in a selected sample. $\alpha = 0.05$ means that above the corresponding threshold value, one can be confident with 95% certainty that the relationship observed is not due to random variation in the sample size. These critical values can be computed, but we use the pre-computed tables for Spearman's rank correlation critical values available online from York University [Uni].

6.2.1 Relationships Between Formal and Graph-Based Measurements in the MCCS

Since we have 10 alternative system designs of the MCCS, only Spearman correlation values with a magnitude of at least 0.564 are considered statistically significant. Each individual design has at most four agents, which poses some difficulty when looking at agent-level measurements since only correlation values with a magnitude of 1.0

can be considered statistically significant [Uni]. In general, we consider a correlation "strong" if the magnitude is at least 0.7.

System-Level Measurements

The Spearman correlation values between the prevalence of implicit interactions and the graph-based measurements for the MCCS design alternatives are given in Table 6.6. Of these, only the relationships between the number of direct interactions, cyclomatic complexity, number of cycles, largest cycle size, median cycle size, and number of intended interactions have a statistically significant relationship with both measurements of implicit interactions. Those graph-based measurements that have statistically significant relationships with the implicit interactions measurements all indicate strong positive relationships, meaning that if these measurements increase from one system design alternative to the next, the prevalence of implicit interactions and number of implicit interactions reliably increases. This tells us that the number of communication channels in a system design for the MCCS is useful for approximating the formal method results, as is the existence and size of cycles in those communication channels.

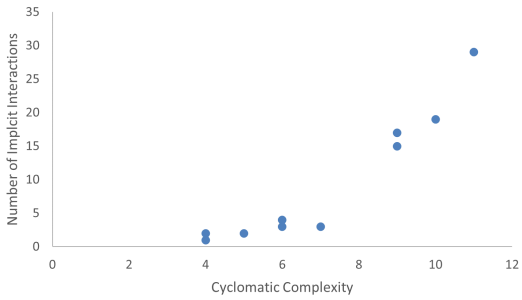
The number of different stimuli sent or the behaviors of the agents in the system are not effective predictors. Given that implicit interactions deal with the *potential* for communication, it makes sense that the structure of the system's communication is more relevant than what is communicated, but it is interesting at how little the size of the C^2KA specification relates to the prevalence of implicit interactions since a greater number of stimuli would suggest more communication between agents in the

Table 6.6: Spearman correlations between the prevalence of implicit interactions in the MCCS alternative system designs and graph-based-measurements

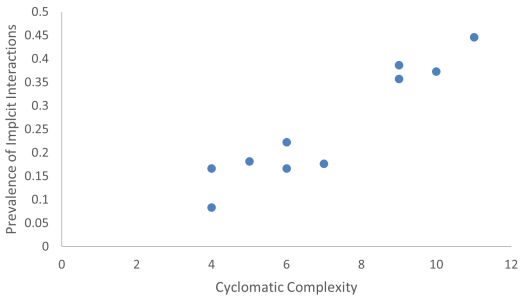
	Number of Implicit Interactions	Prevalence of Implicit Interactions
Number of Agents	0.3503	0.1746
Number of Direct Interactions	0.9783	0.8628
Cyclomatic Complexity	0.96	0.8835
Number of Stimuli	-0.2922	-0.4940
Number of Behaviors	-0.0699	-0.1963
Size of C^2KA Specification	-0.2105	-0.3581
Number of Cycles	0.8792	0.8765
Largest Cycle Size	0.7833	0.8590
Median Cycle Size	0.8792	0.8765
Number of Intended Interactions	0.9632	0.8196
Variance of Degree Centrality	-0.1963	-0.3609

system. It is understandable that we do not see a statistically significant relationship between the number of agents and the prevalence of implicit interactions in this sample, as we do not have design alternatives with a wide range of different numbers of agents. We were able to produce design alternatives with a lower prevalence of implicit interactions which both increased and decreased the variance of agent degree centrality, but we do not see a relationship indicating more centralized or distributed systems being better overall, nor do we find a general trend of the highest and lowest degree variance values always corresponding to the lowest prevalence of implicit interactions.

The similarity in correlation values for the number of direct interactions and cyclomatic complexity make sense when we consider that the number of direct interactions (i.e., the number of edges in the communication graph) is used to calculate cyclomatic complexity [McC76] as shown in Equation 5.5. It also makes a lot of sense that more



(a) Cyclomatic complexity of each system design alternative plotted against the number of implicit interactions in that design

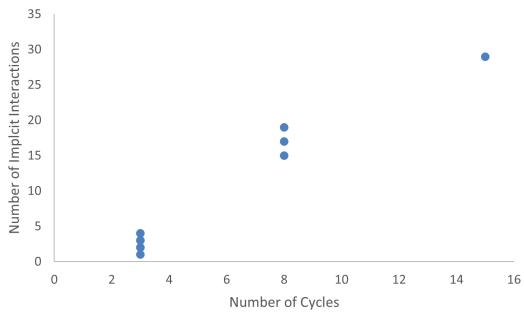


(b) Cyclomatic complexity of each system design alternative plotted against the prevalence of implicit interactions in that design

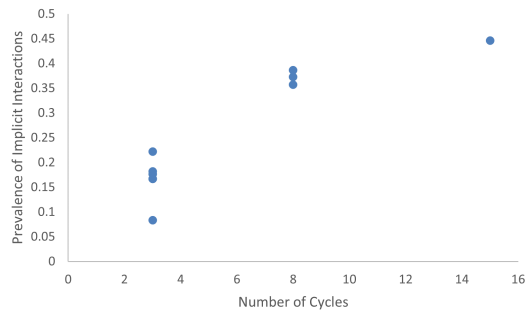
Figure 6.9: The relationship between cyclomatic complexity and the prevalence of implicit interactions in the MCCS

direct interactions correlates with the number of implicit interactions if we consider previous theories that larger systems will have more implicit interactions [JV17b]. A larger system may require more communication channels, but the correlation with the prevalence of implicit interactions is interesting as this is meant to normalize the implicit interactions measurement with respect to system size. The relationship between cyclomatic complexity and the prevalence of implicit interactions in the MCCS is shown in Figure 6.9. Each point on the plots represents one alternate system design. While Figure 6.9b shows a linear relationship between cyclomatic complexity and the prevalence of implicit interactions, Figure 6.9a seems to increase more quickly.

Figure 6.10, which shows how the number of cycles in a system design relates to the prevalence of implicit interactions, tells us that unlike cyclomatic complexity, the number of cycles has a linear relationship with the number of implicit interactions. The data in Figure 6.10a is much more clumped together than Figure 6.9 as several design alternatives have the same number of cycles, but a larger number of cycles



(a) Number of cycles in each system design alternative plotted against the number of implicit interactions in that design

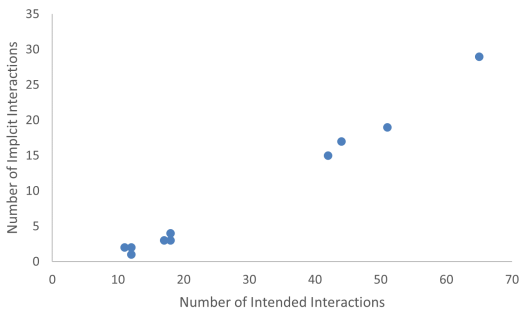


(b) Number of cycles in each system design alternative plotted against the prevalence of implicit interactions in that design

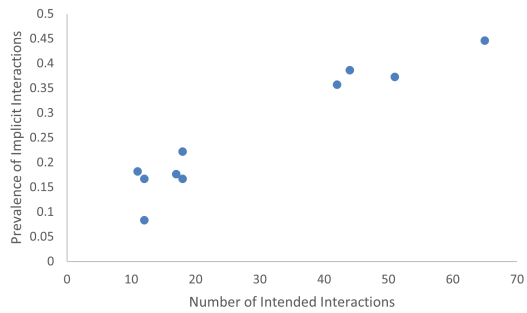
Figure 6.10: The relationship between number of cycles and the prevalence of implicit interactions in the MCCS

still reliably relates to a higher prevalence of implicit interactions. Figure 6.10b is similar. While the data points are grouped around specific numbers of cycles, the sets of alternative system designs with fewer cycles in their communication graphs have a consistently lower prevalence of implicit interactions. Implicit interactions are effectively *paths* in the communication graph [JV17b], meaning they are walks in the graph that do not repeat nodes, so it is interesting to see that all the measurements of cycles, which would allow a walk in the graph to repeat nodes, relate to implicit interaction prevalence within system designs.

The number of intended interactions in a system design also shows a strong positive relationship with both the number of implicit interactions in the system, shown in Figure 6.11a, and with the prevalence of implicit interactions, shown in Figure 6.11b. Both these relationships appear approximately linear.



(a) Number of intended interactions in each system design alternative plotted against the number of implicit interactions in that design



(b) Number of intended interactions in each system design alternative plotted against the prevalence of implicit interactions in that design

Figure 6.11: The relationship between number of intended interactions and the prevalence of implicit interactions in the MCCS

Agent-Level Measurements

Table 6.7 shows the Spearman's rank correlation values between the different node centrality measures taken from each MCCS design alternative's interaction model and the frequency each agent occurs in the design alternative's identified implicit interactions. The latter measurement is equivalent to the numerator of the agent contribution ratio in Equation 5.2. A cell value of "–" indicates that one or both of the sets of values used in the correlation are all equal. A correlation cannot be calculated in these cases because both sets of values need some amount of variance in order to determine the degree to which they vary together.

Though for most design alternatives degree centrality, fan-in, and PageRank all show strong positive relationships, we cannot draw conclusions from this data due to the lack of statistical significance. None of the tested node centrality measures show consistent strong or statistically significant relationships with the agent contribution

Table 6.7: Spearman correlations between node centrality methods and the number of implicit interactions that include each agent in the MCCS

System Design Alternative	Degree Centrality	Fan-In	Fan-Out	Eigenvector Centrality	PageRank
MCCS1	0.9487	0.9487	0	–	0.9487
MCCS2	0.7746	0.5443	0.5443	0.5774	0.7746
MCCS3	0.7746	0.7746	0.5443	0.5774	0.7746
MCCS4	0.7746	0.7746	0.5443	0.5774	0.7746
MCCS5	0.9487	0.9487	0.8889	0.5443	0.9487
MCCS6	0.5443	0.5443	0.5443	0.3333	0.5443
MCCS7	–	–	–	–	–
MCCS8	–	–	–	–	–
MCCS9	0.9487	0.9487	0.8889	0.5443	0.9487
MCCS10	0.9428	0.9428	0.9428	0.5774	0.9428
Number Strong Relationships	7	6	3	0	7
Number Statistically Significant	0	0	0	0	0

ratios in each MCCS design alternative. While we can use the system-level correlations from the MCCS moving forward in Chapter 7, we can only use the agent-level correlations from the WDS due to the limitations set by the MCCS system size.

6.2.2 Relationships Between Formal and Graph-Based Measurements in the WDS

There are 29 design alternatives for the WDS, so we can only consider Spearman correlations with a magnitude of at least 0.312 to be statistically significant for system-level measurements. Design alternatives WDS1 to WDS11 and WDS14 to WDS21 have seven agents, so agent-level measurements must have a Spearman correlation

magnitude of at least 0.714. WDS12, WDS13, and WDS22 to WDS29 all have eight agents, so agent-level measurements for these alternative designs can have a Spearman correlation magnitude as low as 0.643 to be considered statistically significant [Uni]. Like the MCCS, we consider a correlation "strong" if the magnitude is at least 0.7.

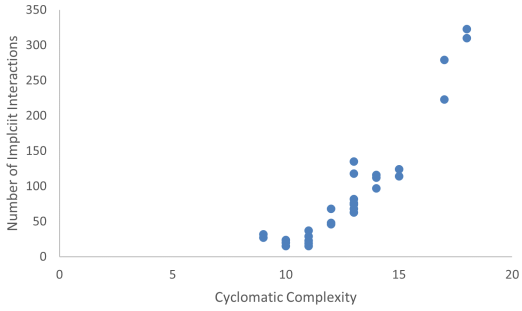
System-Level Measurements

The Spearman correlation values between the system-level implicit interaction measurements and the graph-based measurements for the WDS design alternatives are given in Table 6.8. Of these measurements, the number of direct interactions, cyclomatic complexity, and number of cycles show statistically significant, strong positive relationships with both measurements of implicit interactions, just as they did in the MCCS. This tells us that the number of communication channels and cycles in those communication channels are important for approximating the formal method results since these measurements show similar high correlation values in both sets of system designs.

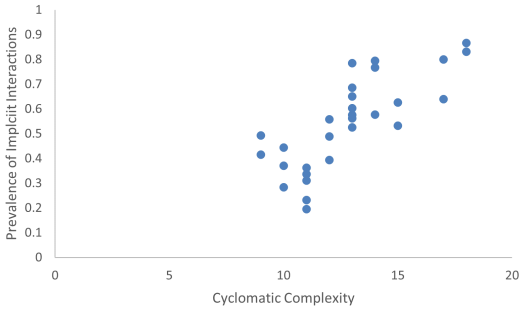
Figure 6.12 shows the relationship between implicit interaction measurements and cyclomatic complexity. In this case, both appear approximately linear, with the wider spread of data in Figure 6.13b corresponding to the lower correlation value between cyclomatic complexity and the prevalence of implicit interactions. Figure 6.13 shows the relationship between implicit interaction measurements and the number of cycles. In the WDS we have much more variety in the number of cycles across different design alternatives when compared to the MCCS, but we see similar shaped trends. The relationship in Figure 6.13a is more linear, while the relationship in Figure 6.13b

Table 6.8: Spearman correlations between the prevalence of implicit interactions in the WDS alternative system designs and graph-based-measurements

	Number of Implicit Interactions	Prevalence of Implicit Interactions
Number of Agents	0.4120	0.3208
Number of Direct Interactions	0.8962	0.8072
Cyclomatic Complexity	0.9166	0.8267
Number of Stimuli	-0.1634	0.0284
Number of Behaviors	0.4575	0.4958
Size of C^2KA Specification	0.0842	0.2163
Number of Cycles	0.8956	0.7843
Largest Cycle Size	0.7468	0.5758
Median Cycle Size	0.5723	0.4406
Number of Intended Interactions	0.1512	0.2929
Variance of Degree Centrality	0.3991	0.5808



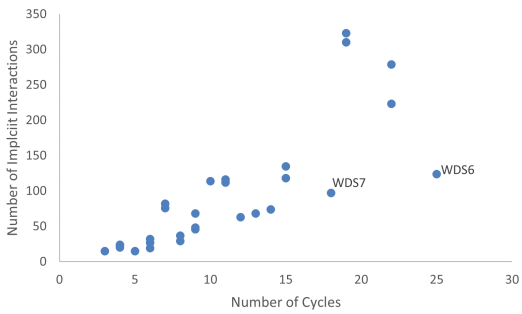
(a) Cyclomatic complexity of each system design alternative plotted against the number of implicit interactions in that design



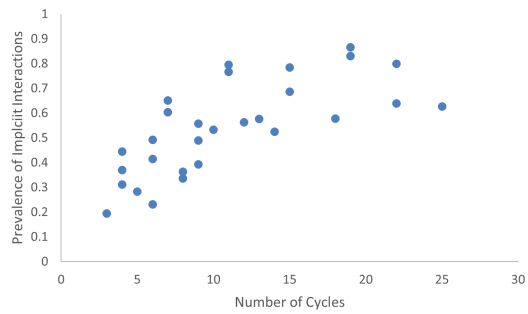
(b) Cyclomatic complexity of each system design alternative plotted against the prevalence of implicit interactions in that design

Figure 6.12: The relationship between cyclomatic complexity and the prevalence of implicit interactions in the WDS

appears more logarithmic. Two notable exceptions to the monotonically-increasing trend in Figure 6.13a are labelled. These are **WDS6** and **WDS7**, which both increase the amount of stimuli broadcast between agents without removing the oversights present in the system. These system designs would be expected to have a particularly



(a) Number of cycles in each system design alternative plotted against the number of implicit interactions in that design



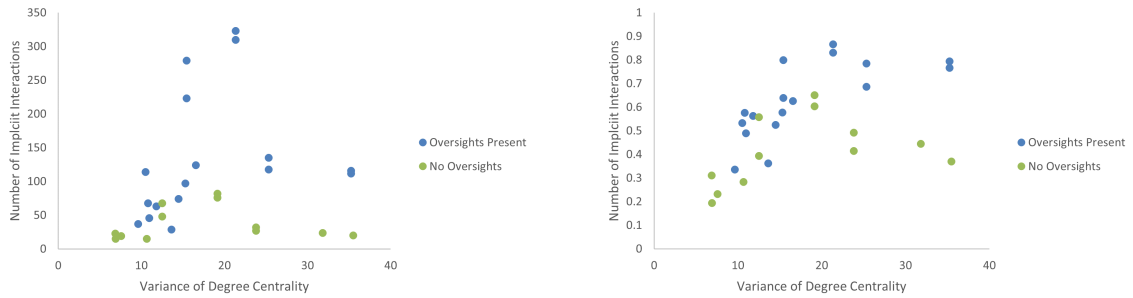
(b) Number of cycles in each system design alternative plotted against the prevalence of implicit interactions in that design

Figure 6.13: The relationship between number of cycles and the prevalence of implicit interactions in the WDS

high number of cycles since the broadcast stimuli are sent in both directions, and both the higher amount of broadcast stimuli and the presence of oversights increase the overall connectivity of the interaction model.

The differences in the strength of relationships for the MCCS and WDS are more interesting. The two measurements related to the size of cycles still show statistically significant relationships with both implicit interaction measurements, though the relationships are not as strong in the WDS. Only the relationship between the largest cycle size and number of implicit interactions is considered strong in the WDS, whereas all four combinations of cycle size measurement and implicit interaction measurement showed strong relationships in the MCCS.

The variance of degree centrality also shows a statistically significant relationship with both measurements of implicit interactions, though it is not a strong monotonic relationship. The relationship between the variance of degree centrality and the measurements of implicit interactions is shown in Figure 6.14. In both Figure 6.14a



(a) Variance of degree centrality in each system design alternative plotted against the number of implicit interactions in that design

(b) Variance of degree centrality in each system design alternative plotted against the prevalence of implicit interactions in that design

Figure 6.14: The relationship between the variance of degree centrality and the prevalence of implicit interactions in the WDS

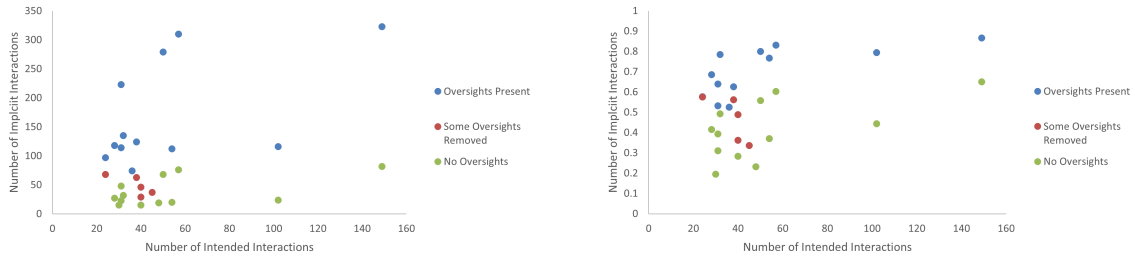
and Figure 6.14b, we see that there is a more clear relationship with the implicit interactions measurements at lower variance values, which starts to diverge as the variance increases further. In both cases the divergence can be explained due to the presence of oversights in the system, indicated by the color of each data point. While for the complexity measures, separating the design alternatives with and without oversights does not alter the apparent trends in the data, in Figure 6.14 it is useful to separate visually whether or not oversights exist in a system design. In both figures, we can see that whether or not oversights are present in the WDS has a greater impact on the implicit interactions present in more centralized system design alternatives (i.e., those design alternatives further to the right of each plot). Unlike in the MCCS, we notice that in the WDS design alternatives without oversights, the designs with the highest and lowest degree variance values have lower implicit interaction values than those in the middle, but this relationship is not observed when considering all design alternatives.

The number of behaviors in the C^2KA specification of the system design has a weaker, but still statistically significant, relationship with both measurements of implicit interactions. This can largely be explained by the design alternatives which increase the number of agents in the system, since any additional agent will require its own additional behavior(s).

It is surprising that the number of intended interactions in the system does not show a relationship with either implicit interaction measurement in the WDS, given how strong the relationship was in the MCCS. This difference is in part also explainable by the presence of oversights. In Figure 6.15a and Figure 6.15b, each design alternative is colored to designate whether of not oversights are present in the design. In these figures, it is also useful to distinguish which design alternatives have some oversights eliminated. These are the design alternatives first listed in Table 5.11. With this distinction, it is easier to see how the presence of oversights increases the overall variation in this data. When recalculating the correlation for just the 12 WDS design alternatives that have no oversights (i.e., those shown in green in Figure 6.15b), the Spearman correlation is now 0.5674, which is considered statistically significant for 12 data points [Uni], but is still not as strong a relationship as seen in the MCCS.

Agent-Level Measurements

Table 6.9 shows the Spearman's rank correlation values between the different node centrality measures taken from each WDS design alternative's interaction model and the frequency each agent occurs in the design alternative's identified implicit interactions. This is the same comparison as Table 6.7, but for the WDS rather than the



(a) Number of intended interactions in each system design alternative plotted against the number of implicit interactions in that design

(b) Number of intended interactions in each system design alternative plotted against the prevalence of implicit interactions in that design

Figure 6.15: The relationship between number of intended interactions and the prevalence of implicit interactions in the WDS

MCCS. Each row represents one of the 29 WDS design alternatives, and each column represents one centrality method.

Table 6.9 tells us that, in general, higher node centrality values tend to indicate a higher number of implicit interactions that travel through an agent. We include in Table 6.9 how many of the design alternatives show statistically significant relationships between the formal method results and node centrality methods, and how many of those relationships are strong. With the exception of PageRank, most show strong, statistically significant relationships in the majority of the alternate designs. The local node centrality measures of fan-out and degree centrality best approximate which agents appear most frequently in the system's implicit interactions. If a global influence centrality measure is preferred, eigenvector centrality should be used.

Unfortunately, like the MCCS, none of the tested node centrality measures show consistent statistically significant relationships with the agent contribution ratios in the WDS, meaning we still require the formal analysis if we want to know exactly

Table 6.9: Spearman correlations between node centrality methods and the number of implicit interactions that include each agent in the WDS

System Design Alternative	Degree Centrality	Fan-In	Fan-Out	Eigenvector Centrality	PageRank
WDS1	0.9909	0.867	0.9167	0.9722	0.887
WDS2	0.9636	0.6608	0.9909	0.9909	0.6732
WDS3	1	0.9063	0.9542	0.963	0.7106
WDS4	1	0.676	0.9816	0.963	0.5169
WDS5	0.6606	0.3965	0.7364	0.6973	0.4488
WDS6	0.9167	0.9907	0.8808	0.8981	0.8808
WDS7	0.8889	0.9907	0.8889	0.8981	0.8808
WDS8	0.8909	0.8119	0.8909	0.9358	0.7455
WDS9	0.8545	0.8874	0.7707	0.8808	0.7818
WDS10	0.789	0.7175	0.7707	0.789	0.4182
WDS11	0.6944	0.705	0.75	0.75	-0.1101
WDS12	0.8	0.5468	0.8313	0.7286	-0.4699
WDS13	0.897	0.7323	0.8836	0.7831	0.2909
WDS14	0.9542	0.8602	1	0.9909	0.8602
WDS15	0.9909	0.8975	0.9909	0.9342	0.8975
WDS16	0.8909	0.6362	0.9636	0.963	0.6362
WDS17	0.9358	0.8748	0.8808	0.6179	0.8748
WDS18	0.9542	0.8602	1	0.9909	0.8602
WDS19	0.9542	0.8602	0.9909	0.9342	0.8602
WDS20	0.8909	0.6362	0.9636	0.963	0.6362
WDS21	0.8909	0.5394	0.9358	0.6179	0.5394
WDS22	0.8232	0.7268	0.744	0.7827	0.4207
WDS23	0.9759	0.9262	0.8768	0.9184	0.7108
WDS24	0.9329	0.8635	0.819	0.9691	0.628
WDS25	0.9759	0.9262	0.8768	0.9184	0.7108
WDS26	0.8072	0.5396	0.8485	0.7108	0.4243
WDS27	0.994	0.8736	0.994	0.8768	0.6909
WDS28	0.9572	0.7851	0.9383	0.9572	0.6667
WDS29	0.9693	0.7998	0.9939	0.8626	0.589
Number Strong Relationships	27	21	29	26	13
Number Statistically Significant	27	20	29	26	14

those agent contribution ratios. This is okay, as we use the agent contribution ratios primarily to help generate the alternate system designs that we are using to identify relationships, formal approximations, and design principles. We are not arguing that contribution ratios should always be used in all system development. Agent weak points from the contribution ratios are only relevant if a later design has a formal model, which we are aiming to reduce reliance on through this work by providing alternate strategies.

We have seen in Section 5.2 that contribution ratios do not necessarily point to the same areas to focus redesign efforts as the frequency that each agent occurs in the system design's implicit interactions, but the latter's relationship with node centrality methods means we still have an area on which to focus redesign without applying formal methods. Rather than approximating which agents contribute *relatively* the most to the implicit interactions in a system given the agents' contribution to system objectives, the node centrality measures can approximate which agents contribute *absolutely* the most to the implicit interactions in a system.

6.2.3 Combining Measurements in a Multiple Regression Model

Given that multiple system-level measurements seem to be useful in gauging how the measurements of implicit interactions change from one system design to another in both the MCCS and WDS, we experiment to see if multiple measurements can be used together to produce greater predictive power. The primary goal of this experiment is not simply the construction of a model to predict the prevalence of implicit interactions in a system design, but to show that by considering more than

one measurement that relates to the existence of implicit interactions, we can better understand whether there will be an increased or decreased prevalence of implicit interactions from one system design alternative to the next without performing a full formal analysis of the system designs.

A multiple regression model is a regression model that uses multiple independent variables to predict a dependent variable, given by the linear equation:

$$\hat{y}_i = \beta_0 + \beta_1 x_{1i} + ... + \beta_n x_{ni} \tag{6.1}$$

where x_1, ... x_n are the n independent variables used, and β_1, ... β_n are coefficients tuned given a dataset of dependent variable samples y and independent variable samples x [Sah16]. We score regression models with the *coefficient of determination* (r^2), which describes how much of the variation in a dependant variable can be explained by one or more independent variables. For example, an r^2 value of 0.75 means 75% of the variation in the dependent variable can be explained by the variation in the independent variable(s). For a single independent variable, the r^2 value is just the square of the Pearson correlation value. In general, the r^2 value can be computed with the following equation:

$$r^2 = \frac{\sum (y_i - \bar{y})^2 - \sum (y_i - \hat{y}_i)^2}{\sum (y_i - \hat{y}_i)^2} \tag{6.2}$$

where y_i is the i-th dependent variable value, $\bar{y}$ is the average of all dependent variable values, and $\hat{y}_i$ is the i-th dependent variable value predicted by the i-th independent variable value(s) [Sah16].

With multiple graph-based measurements that are related to the prevalence of implicit interactions in our system designs, we can use a multiple regression model to combine those graph-based measurements to see if together they more fully explain the variation in prevalence of implicit interactions in the system than the individual measurements. In addition, by varying which graph-based measurements are used as the independent variables in our regression model, we can see how much the coefficient of determination changes to compare the prediction strength of individual measurements. The independent variables that, through exclusion, cause the largest drop in the regression model's r^2 value are the independent variables that best describe the variation in the dependent variable [Sah16]. In our context, this means that the measurements we exclude that cause the greatest drop in the regression model's r^2 value are those that best approximate the prevalence of implicit interactions in the system that we would otherwise have to identify through formal analysis. Since this is our motivation for the regression analysis, we present only the r^2 value from each regression model and do not include the coefficients for every model constructed.

For the regression model, we want to combine all measurements that are shown to have a strong, statistically significant relationship with at least one of the measurements of implicit interactions. To perform this analysis, however, we need to assume the measurements are independent and the relationship with each individual measurement is linear. Though many of the plots presented pass a visual check for linearity, we use Spearman's rank correlation when looking at individual measurements because we are primarily interested in whether an increase in a graph-based measurement coincides with an increase in implicit interaction measurements, not whether that increase is linear, exponential, or otherwise. Since the r^2 value of a regression model

with one independent variable is just the square of the Pearson correlation value, which measures the strength of linear relationships, we create regression models using each graph-based measurement individually to check data linearity. We also know that not all measurements are independent, so we exclude some. For example, the number of agents and direct interactions are used to calculate cyclomatic complexity [McC76], and we have two measurements related to the size of cycles. For this reason, we exclude the number of agents, number of direct interactions, and median cycle size from the regression analysis. We exclude median cycle size and not largest cycle size because the latter measurement shows a strong relationship across both system design alternatives, whereas median cycle size shows weaker relationships in the WDS.

Multiple Regression Models for the MCCS

In Table 6.10, we show the results of performing a multiple regression analysis with cyclomatic complexity, number of cycles, median cycle size, and number of intended interactions as independent variables, and each measurement of implicit interactions as dependent variables. We create a regression model that includes all four graph-based measurements, regression models which each exclude one of the four graph-based measurements, and regression models that only include one of the four graph-based measurements. The results reported are the r^2 values for each model, indicating which sets of independent variables have the greatest predictive power.

The first row includes that, collectively, these four graph-based measurements explain 98.72% of the variance in the number of implicit interactions and 94.7% of the

Table 6.10: r^2 values for regression models with different subsets of independent variables for the MCCS

Independent Variables Used	Number of Implicit Interactions	Prevalence of Implicit Interactions
All 4 independent variables	0.9872	0.947
Exclude 'cyclomatic complexity'	0.9859	0.9464
Exclude 'number of cycles'	0.8841	0.9182
Exclude 'largest cycle size'	0.9862	0.9239
Exclude 'number of intended interactions'	0.9862	0.9263
Only 'cyclomatic complexity'	0.8835	0.8963
Only 'number of cycles'	0.9595	0.8166
Only 'largest cycle size'	0.7151	0.7852
Only 'number of intended interactions'	0.39	0.4042

variance in the prevalence of implicit interactions in the ten MCCS design alternatives. For both dependent variables, the r^2 values are still quite high when any one of the independent variables are excluded, or when only one of cyclomatic complexity, number of cycles, or largest cycle size are used. The largest decrease in the r^2 value for either dependent variable from excluding an independent variable comes from excluding the number of cycles from the regression model. We do however see the trend that we are looking for: the use of more graph-based measurements produces a stronger relationship with the results of the formal analysis than considering only one measurement.

Multiple Regression Models for the WDS

In Table 6.11, we show the results of performing the same multiple regression analysis for the WDS. Here we only use cyclomatic complexity, number of cycles, and largest

Table 6.11: r^2 values for regression models with different subsets of independent variables for the WDS

Independent Variables Used	Number of Implicit Interactions	Prevalence of Implicit Interactions
All 3 independent variables	0.8655	0.6265
Exclude 'cyclomatic complexity'	0.642	0.5158
Exclude 'number of cycles'	0.8605	0.6066
Exclude 'largest cycle size'	0.8604	0.6194
Only 'cyclomatic complexity'	0.859	0.6065
Only 'number of cycles'	0.642	0.5145
Only 'largest cycle size'	0.4158	0.3114

cycle size as independent variables as the number of intended interactions does not show a relationship with the implicit interactions measurements across the WDS design alternatives. Like Table 6.10, each cell in Table 6.11 shows the r^2 value for a regression model created from the independent variables described by the row title and the dependent variable described by the column title.

The three graph-based measurements together explain 87% of the variation in the number of implicit interactions and 62.17% of the variation in the prevalence of implicit interactions across the 29 WDS design alternatives. We can directly compare these results to the fifth row in Table 6.10, as they use the same independent and dependent variables, to see that three graph-based measurements in the WDS do not as strongly predict changes in the implicit interaction methods as in the MCCS.

The presence of oversights and multiple interfering processes are the two main differences between the MCCS and WDS. Table 5.14 includes the number of oversights present in each system design alternative, so we can add this to the regression model

to see how much of a role oversights play in the decreased predictive power. The resulting regression models, now with four independent variables, have an r^2 value of 0.8664 with the number of implicit interactions as the dependent variable and 0.6636 for the prevalence of implicit interactions. The values are still much lower than the results we see with the MCCS, leaving the varying amounts of interference between the processes in each system design, for which we do not have a single measurement to characterize, as the most likely cause for this difference.

In the WDS, the largest decrease in the r^2 value in Table 6.11 for both dependent variables comes from excluding cyclomatic complexity from the regression model. The decrease in r^2 is also larger than what we see for the MCCS in Table 6.10. We again see that considering multiple graph-based measurements results in a greater predictive power than any one measurement, though the effect is very small in the WDS with the number of implicit interactions as the dependent variable.

6.3 Conclusions

The purpose of this chapter was to compare design alternatives for the MCCS and WDS with each other to understand what design changes impact the formal methods results, and to compare the formal and graph-based measurements to identify what measurements have strong relationships with the formal methods results. In doing so, we can now focus on the design changes and graph-based measurements that are found to relate to implicit interactions, rather than rely on formal methods-based activities. Specifically, the main takeaways from this chapter are:

1. Not every type of design change that successfully mitigates implicit interactions in one system will work in another system.

2. Both sets of system design alternatives show that the number of direct interactions, cyclomatic complexity, number of cycles in the system design communication graph can help to understand which design alternatives with have more implicit interactions.

3. Centrality methods, specifically fan-out, degree centrality, and eigenvector centrality, can be used to approximate which agents in a system design will contribute most to the system's implicit interactions.

In the next chapter, we use the identified relationships with the graph-based measurements in Table 6.6 through Table 6.9 and the design comparisons made in Section 6.1 to determine which system properties and design principles impact the prevalence of implicit interactions in a system design.

Chapter 7

Data Interpretation Activities

In this chapter, we detail the activities presented in Chapter 3 in which we interpret the results of the data analysis performed in Chapter 6. We begin in Section 7.1 by mapping the relationships between formal and graph-based analyses identified in Section 6.2 to system properties and relate these properties to relevant system quality attributes to answer **RQ3.**. In Section 7.2, we identify design principles that relate to the identified system properties and from the specific design comparisons made in Section 6.1 to answer **RQ4.**.

7.1 Identifying System Properties

In Section 5.4.2, we introduced a number of measurements within the context of different system properties. With the analysis results from Chapter 6, we can now

revisit these system properties for a discussion on how they contribute to the existence of implicit interactions in a system design.

Complexity

The structural complexity of a system design has been shown to relate highly to the both the number and prevalence of implicit interactions in a system design. In both the MCCS and WDS, the positive relationships between implicit interaction measurements and the number of direct interactions, cyclomatic complexity, number of cycles, and size of cycles shown in Table 6.6 and Table 6.8 all point to higher complexity coinciding with a higher prevalence of implicit interactions and a larger total number of implicit interactions.

The number of messages sent in system models [KB02, PS12, AFS12], number of global data structures [RD05, AFS12], and number of communication channels [BD02, SB13, RD05, ZT06] have been proposed as measures of complexity in other works, which in our context correspond to the number of direct interactions via stimuli, number of direct interactions via shared environment, and total number of direct interactions, respectively. Works exist tying cycles to complexity, most notably in the dependency cycles of both models and developed systems [ZN07, Oye15]. The centrality methods we explore have also been proposed in many works to characterize complexity, particularly fan-out [RD05, YR92, MKPS00] and degree centrality [RD05, PS12, MKPS00], which both show consistent strong positive relationships with the number of implicit interactions at an agent level in Table 6.9.

Intuitively, it makes sense that larger, more complex system will have more implicit interactions, and is an idea that has been posited in previous works related to implicit interactions [JV17b, JV17a, Jas20a]. A larger, more complex system in general simply has more agents or potential interactions between agents that need to be accounted for in order to address and mitigate the existence of implicit interactions in the system design. We see this with the inclusion of the FAULTMNGR agent in the WDS. Though it helps to eliminate many of the current implicit interactions in the system, the additional agent results in a net increase in implicit interaction measurements in Table 6.2 because there are so many new potential interactions between agents that must be considered. The presence of oversights is also emblematic of this issue, as their presence effectively increases how many direct interactions are in the system, and therefore the complexity of the system, without contributing to the system's objectives. Dealing with size and complexity is one of the major challenges faced in providing adequate security assurance at the time of writing, as a complete view of all agents and possible interactions is often out of reach for large systems [Jas20c]. While a system designer needs to do their best to account for every possible vulnerability, an attacker only needs to find one vulnerability (e.g., an implicit interaction they can exploit) to potentially have a major impact on that system's ability to achieve its objectives.

In the broader context of system design, managing complexity is also a major goal. Cyclomatic complexity was originally developed as a method to characterize the complexity of program control flows, and McCabe states that the cyclomatic complexity for any software module should be kept as low as possible, and suggests

refactoring when a module's cyclomatic complexity surpasses 10 to better manage the complexity [McC76]. Parnas and Weiss state that designs should be as simple as possible, but no simpler, and that designs should be well structured, which they define as being consistent with design principles [PW87]. Complexity also often is detrimental to many software quality attributes. As complexity increases, many works describe how software quality attributes suffer, including flexibility [BD02], modifiability [SB13, ZT06, GPM04], reusability [BD02, ZT06, RD05], maintainability [ZT06, MKPS00, FMM94], extendibility [BD02, ZT06], and understandability [RD05, GPM04]. This is good for our specific context of implicit interactions, because it means that working towards existing desirable goals when designing systems, like managing complexity or improving specific quality attributes, is in line the goal of reducing the prevalence of implicit interactions in system designs.

Coupling

Coupling between agents is often considered during system development, but many works describe coupling as just a specific way to measure complexity [BD02, SB13, ZT06, MSA13]. Coupling in a system design has been expressed through measurements analogous to the number of direct interactions, such as number of communication channels [BD02, SB13, ZT06] and number of dependencies between system components [KM11], which we have seen strongly correlates to both implicit interaction measurements in both sets of system design alternatives in Table 6.6 and Table 6.8. At the agent-level, degree centrality, which we have found to consistently

relate strongly to the number of implicit interactions in Table 6.9, has also been used to measure coupling [BD02, ZT06, SB13, KB02].

Much of the discussion on both the MCCS and WDS design alternatives in Section 6.1 revolves around coupling. Specifically, in the MCCS most subsequent designs based purely on weak point identification repeatedly decouple non-control agents from each other. The communication via shared environment from the handling agent H and processing agent P both referencing `status` are particularly interesting as they provide communication redundancy to ensure the storage agent S is in the correct state for different actions in the system. Redundant communication in general can help to improve reliability [Bea91] and availability [EOS09], but we see in Section 5.3.1 that whenever the `status` communication via shared environment exists, it is a weak point in the system. This points to redundant communication, and therefore the quality attributes that use redundancy as an improvement strategy, as being counterproductive to reducing the number or prevalence of implicit interactions in a system design. We can also intuit this by recognizing that redundant communication channels means the system has overall more communication channels and therefore more coupling and complexity, which both positively correlate with the implicit interaction measurements.

Since coupling has been discussed as a way to characterize complexity, reducing coupling also helps to improve similar quality attributes as reducing overall complexity. Counter to the many quality attributes that benefit from reduced coupling and complexity, less coupling is often detrimental to performance [DAD+04]. This is because greater coupling often results in more direct communication, which has less overhead

than indirect communications propagated through multiple agents. Therefore, re-designing to reduce the number of implicit interactions in a system design may have negative side-effects for performance.

Centralization

Despite our expectation that more centralized design alternatives would have more implicit interactions, as described in Section 5.4.2, we do not see strong evidence of this relationship in the system design alternatives. There is no statistically significant relationship between the variance in degree centrality and the number of implicit interactions in the MCCS or in the WDS when we only consider the design alternatives without oversights. A statistically significant relationship does exist when we include the WDS design alternatives with oversights, though it is not as strong as what we see with other measurements. Given that increased centralization overall decreases system-wide coupling in the MCCS, whereas overall coupling is increased through the centralization in the WDS, the positive relationship we see between centralization and implicit interaction measurements in the WDS may just be due to that increased coupling and complexity. This would also be in line with why we only see a statistically significant relationship when the design alternatives with oversights are included in the set of alternate designs used, because they also increase the coupling and complexity of a design while increasing both measurements of implicit interactions.

Concurrency

As discussed in Section 5.4.2, systems with more concurrency were expected to have more implicit interactions because of the increased communication needed to synchronize their concurrent behaviors. We see in the MCCS that a greater number of intended interactions, which increases with the amount of control flow forks and synchronization required, shows a strong positive relationship with the prevalence of implicit interactions and number of implicit interactions. In the WDS, we only see a relationship between the number of intended interactions and implicit interaction measurements when we ignore the system design alternatives that contain oversights. This makes sense since oversights are, by definition, not information captured by the intended interactions of the system design. The relationship seen between the number of intended interactions and implicit interaction measurements is still not as strong in the WDS design alternatives than in the MCCS design alternatives, but the WDS does contain additional concurrency in the form of interference between the pump-controlling and fault-managing control flows. We see with the MCCS in Figure 6.11 that the relationship between the number of intended interactions and implicit interactions in a system that contains a single main control flow appears linear, but in Section 5.3.2 we have seen how quickly the number of implicit interactions grows when combining multiple control flows together. It appears that both concurrency from multiple agents within one control flow and concurrency from having multiple control flows in a system contribute the number of implicit interactions in a system, but the latter contributes to a much greater extent so we see a weaker relationship with measurements that characterize the former in the WDS.

Incorporating concurrency into a control flow in a system design is one of the main strategies used to improve system performance [DAD+04] since we can have independent actions like those performed by the sample flow meter agent SFM and SO_3 analyzer agent SO3 in the WDS performed in parallel to save time. We can see this visually in the intended interactions of systems, where the event traces are longer for system designs without concurrency (e.g., MCCS6 in Figure 6.4c) when compared to designs with concurrency (e.g., MCCS10 in Figure 6.5b) since without concurrency, only one agent can perform its actions at a time. Since we see a relationship between concurrency and more implicit interactions when a system design contains a single main control flow, like we discuss with coupling, redesigning systems to reduce the number of implicit interactions may be detrimental to system performance, as is often the case with improving security in general [Hen01, Zor94, GCAW07, KCT+20].

7.2 Identifying Design Principles

We now identify design principles from different facets of software and system design that either relate to the identified system properties in Section 7.1 or capture the spirit behind specific design changes discussed in Section 6.1. The identification of these design principles, and specifically how each relates to the prevalence of implicit interactions, can provide guidance to system designers, enabling them to avoid the need for time-consuming formal analyses to improve system security with respect to these vulnerabilities at each iteration of the design process.

Low Coupling and Law of Demeter

Greater coupling at a system-level through the number of direct interactions has been shown in Section 6.2 to relate highly to the prevalence of implicit interactions. Since we discuss in detail in Section 7.1 how more coupling relates to more implicit interactions, the principle *Low Coupling* is relevant. This principle states that systems should be designed so that they minimize the degree to which each component depends on others in the system [QFTX10].

Coupling also relates highly to the *Law of Demeter*, which states that designers should limit the knowledge each component has about other components, and to only allow components to communicate with others that are closely related [Lie89]. In the MCCS, the decoupling of P and S discussed in Section 6.1.1 is particularly relevant here because H is the agent that actually transfers material from S to P, so we do not necessarily need P and S to directly communicate, and removing this direct communication causes both implicit interaction measurements to decrease. For the pump-controlling process within the WDS, the actions of agents SAP, SFM, and SO3 do not need to know any information about PLC, and so in the original design the PLC only communicates with these agents when faults are resolved. When the WDS design is centralized (e.g., **WDS15**), PLC now needs to communicate directly to these three agents and the prevalence of implicit interactions is increased.

High Cohesion and Single Responsibility

We discuss in Section 6.1.1 how many of the design changes made by transforming the MCCS around weak points improve cohesion in the system while reducing the prevalence of implicit interactions, which relates to the principle of *High Cohesion*. This principle states that components should be designed so that they perform a set of closely related operations [QFTX10]. This is very similar to the concept of *Single Responsibility*, which describes how a component should only have one responsibility, and is often expressed as the idea that a component "should have only one reason to change" [MNK03].

Ideally, if the MCCS has a control agent, then it should have the responsibility of controlling system operations. The other agents should perform actions related only to what their name suggests: storing material for S, handling material for H, and processing material for P. In design alternatives like **MCCS6** that have a lower prevalence of implicit interactions, only C needs to change when changes to the overall control flow are made because each agent has a single responsibility, whereas and changes to the control flow in **MCCS1** may require changes to agents beyond just the control agent as several agents take part in coordinating actions.

In the WDS, the addition of the fault manager agent FAULTMNGR at first seems to counter the relevance of the principles of high cohesion and single-responsibility because both the implicit interaction measurements increase. However, we have seen in Section 6.1.2 how the addition of FAULTMNGR does successfully eliminate implicit interactions in the system while improving overall cohesion. In this case, it appears that while relevant, cohesion may not have as much of an impact on the prevalence

of implicit interactions as the impact of size and coupling. It may be the case that improving cohesion through reorganizing responsibilities as we do in the MCCS is effective at reducing the number of implicit interactions, while improving cohesion through splitting responsibilities into multiple agents as we do in the WDS is simply offset by the extra communication required for an additional agent to be included in the system.

Isolation and Modularity

It is understandable that *Low Coupling* and *High Cohesion* relate to a lower prevalence of implicit interactions, as other works have expressed how low cohesion or too much coupling makes the overall structure of the system hard to manage, expand, maintain, and modify, leading to more complexity and susceptibility for unwanted component interactions which can propagate unexpected behaviors through the system [Per99]. Proper cohesion and coupling are also required to achieve proper *Isolation* and *Modularity* [SB18]. The principle of Isolation states that public access systems should be isolated from critical resources to prevent disclosure or tampering, while the principle of Modularity relates to the use of modular architectures and the development of security functions as separate, protected modules. A more isolated agent is less coupled in the system so there should be fewer ways for the agent to be influenced through implicit interactions. A system designed into proper modules is also better than all system behavior placed within a single large agent. We do not explicitly create a model for this, because a system with one agent would have no implicit interactions as there would be no communication between different

agents. However, this does not mean that the implicit linkages between actions no longer exist, they simply are no longer at the design level from which we perform our analysis. Furthermore, a single large agent, sometimes referred to as a "god object," has several additional issues, including low cohesion, maintainability, and understandability [Fow18].

Least Common Mechanism and Open-Closed Principle

Least Common Mechanism is a security design principle which states that mechanisms used to access resources should not be shared [SS75]. Sharing resources provides a channel along which information can be transmitted, and so unnecessary resource sharing should be avoided. When considering specific system properties, a greater amount of resource sharing could be deemed necessary to facilitate reliability, adaptation, or recovery, but in a security context these additional communication channels provide additional vectors an attacker may be able to exploit. In a general software engineering context, we often minimize shared environments such as global variables as much as possible since they are typically seen as bad design practice [QFTX10]. Moreover, least common mechanism has been described as a principle that "helps reduce the number of unintended communication paths," [SB18] which is very relevant to the focus of this work. Resource sharing is also captured in the *Open–Closed Principle*, which states that a component should be open to extension, but closed for modification [Mar00]. This relates to encapsulation, an object-oriented specific version of Isolation [SB18], but the Open–Closed Principle also discourages the use of global variables. The more global variables or shared environments in a system, the

more coupling and complexity exist, which both relate to the prevalence of implicit interactions. We see in several design alternatives how shared environments often have high contribution ratios and that when two agents communicate with each other by both stimuli and shared environment, the shared environments frequently appear as interaction weak points over the communication via stimuli in both the MCCS (e.g., **MCCS1** in Figure 5.1, **MCCS2** in Figure 5.3a, **MCCS5** in Figure 5.5) and the WDS (e.g., **WDS1** in Figure 5.2, **WDS4** and **WDS11** in Figure 5.9, **WDS13** in Figure 5.10).

Economy of Mechanism

With the clear relationship established with complexity in Section 7.1, *Economy of Mechanism* [SS75] is highly relevant to the prevalence of implicit interactions within a system design. Economy of Mechanism promotes the idea that systems should be as simple and small as possible to reduce the opportunities for an attacker to discover vulnerabilities to exploit. We have shown that more complex and larger systems have a higher prevalence of implicit interactions through relationships identified across all design alternatives for both the MCCS and WDS. We have also seen specific examples in how making the system simpler and smaller through the elimination of both stimuli (e.g., Section 6.1.2) and shared environments (e.g., Section 6.1.1 and Section 6.1.1) reduces the prevalence of implicit interactions.

It is important to note that eliminating communications, of course, may compromise system objectives such as the added reliability that the redundancy through the `status` variable in the original MCCS provides. Redundant communication may be

seen as necessary in a system design depending on system requirements, so while eliminating communication such as `status` may make the system more *simple and small*, it may not be viewed as a *possible* design change. Oversights on the other hand, as seen in the WDS, are communications that can be eliminated to make a system smaller without compromising system objectives, as we discuss in Section 7.1.

Separation of Privilege

The MCCS design alternatives centered around the `status` shared variable are interesting when we consider *Separation of Privilege*, which describes how a system should not grant permission based on a single condition [SS75]. In the original manufacturing cell design (**MCCS1**), when `status` is present, it is used by H and P to ensure that S is ready for certain actions to be performed by C issues commands. In this case, H and P have two conditions on their actions: (1) the reception of stimuli from C and (2) the correct value of `status`. Since multiple conditions are requiring more communication between agents, granting permission based on more conditions is worse for the prevalence of implicit interactions in system designs.

7.3 Conclusions

The purpose of this chapter was to use the analysis conducted in Chapter 6 to identify system properties and existing design principles that are relevant to mitigating the prevalence of implicit interactions in system designs through exploring literature

related to system properties, design principles, and the identified graph-based measurements that have shown a relationship to implicit interaction measurements. This provides yet another layer of abstraction from the formal-methods based activities, in which we can now focus on conformance to design principles in order to reduce the number or prevalence of implicit interactions. The main takeaways from this chapter are:

1. Given our data collected in Chapter 5 and the relationships identified in Chapter 6, the size and complexity of systems is the most impactful system property for increasing the number of implicit interactions.

2. In systems with a single main control flow, the amount of concurrency between agent actions plays a role in the number of implicit interactions. When there are multiple control flows in a system, interference between those control flows has a larger impact on the number of implicit interactions.

3. Adherence to fundamental software design principles like Low Coupling and High Cohesion [QFTX10] are in line with the goal of mitigating implicit interactions in a system design.

4. The application of well-known security design principles [SS75, SB18] are also found to be in line with the goal of mitigating implicit interactions, with the important exception of Separation of Privilege when multiple privilege attributes are brought together from different parts of the system design.

The remaining chapters contain a broad discussion of many aspects of this work, including the applicability of the findings, threats to validity, and possible future works.

Chapter 8

Discussion

In this chapter, we explore the strengths and limitations of the approach detailed in Chapters 5, 6, and 7. In Section 8.1, we focus on the additional formal analysis that builds on existing works [JV17b] and answers research question **RQ1.**, described and demonstrated in Chapter 5. In Section 8.2, we focus on the graph-based measurements we collect which answer research question **RQ2.**, identified in Chapter 6. In Section 8.3, we focus on the system properties, quality attributes, and design principles which answer research questions **RQ3.** and **RQ4.**, identified in Chapter 7. In Section 8.4 and Section 8.5, we focus on the limitations of the research approach and threats to the validity of the results obtained.

8.1 Strengths of Formally Identifying System Weak Points

Research question **RQ1**. is motivated by the results of a questionnaire distributed to domain experts of the WDS in previous work [Jas20b] to understand the usefulness of the implicit interaction identification presented in Section 5.1 for improving system security. While every respondent agreed that the results were understandable and valuable, individual feedback included comments raising the issue that the presentation of the implicit interactions in a form like Table 5.2 requires too much expertise, and that it would be more beneficial to provide a summary of problematic areas and mitigation advice. Using the contribution ratios described in Equation 5.2 and Equation 5.3 and identifying weak points helps to make all this information much more easily digestible and provides more actionable results to directly address the suggestions for improvement made by domain experts.

Beyond simply addressing previous critiques, the ability to prioritize vulnerabilities is important as it is often not possible to address every vulnerability present in the system. With our focus on implicit interactions, this means requiring some method to determine which aspects of the system contribute most their existence, so we can most effectively mitigate them given the available resources. A frequency analysis of the agents and direct interactions that exist in the system's implicit interactions is a simple and effective method to achieve exactly this, and identifying both agent and interaction weak points provides multiple views of the problem that can paint a more complete picture as to why the identified weak points are problematic in the system.

Since this prioritization approach is driven solely by the characteristics of the implicit interactions, it avoids any subjectivity in the results from the practitioner's input. A data-driven approach such as this improves objectivity, can be easily automated, and provides a more reproducible and consistent outcome.

8.2 Strengths of Identifying Relevant Graph-Based Measurements

The benefits of identifying graph-based measurements with relationships to the implicit interaction measurements we collect are twofold. First, we are able to tie implicit interactions to system properties and design principles in Chapter 7, since there is a much greater volume of work on the relationship between system properties and measurements taken from systems than there is work on implicit interactions. By relating implicit interactions to other system measurements, we are able to relate implicit interactions to system properties based on what properties other works characterize with the same measurements. Second, these measurements can be used directly to approximate the formal analysis results, allowing us to reason about whether the prevalence of implicit interactions changes from one design to the next without actually performing the formal analysis.

These benefits enable a wider range of practitioners to consider the security threat implicit interactions pose without requiring the formal methods expertise to use existing works. One common critique of formal methods are scalability issues [Hei98] and so using the graph-based methods presented to track implicit interactions across

Table 8.1: Execution times to calculate formal and graph-based measurements for each MCCS design alternative

System Design Alternative	Formal Methods Execution Time	Graph-Based Measurement Execution Time
MCCS1	12.9647s	0.0045s
MCCS2	13.7693s	0.0038s
MCCS3	14.9017s	0.0038s
MCCS4	14.2747s	0.0042s
MCCS5	17.5392s	0.0072s
MCCS6	17.7228s	0.0076s
MCCS7	5.7559s	0.0032s
MCCS8	5.6447s	0.0031s
MCCS9	16.1764s	0.0073s
MCCS10	16.1414s	0.0066s
Total Execution Time	134.8914s	0.0516s
Average Execution Time	13.4891s	0.0052s

design alternatives, like simply counting the number of communication channels at a system or agent-level, opens up implicit interactions as a design concern that can be analyzed in larger systems. On a machine with a 2.6 GHz Intel Core i7 processor and 8 GB RAM, the completion times for the formal analysis and the graph-based approximations are given in Table 8.1 for the MCCS design alternatives and Table 8.2 for the WDS design alternatives, highlighting the speedup that can be gained through relying on graph-based approximations during an iterative design phase.

In the case of the WDS, formally evaluating all 29 design alternatives takes over 23 minutes, whereas calculating the graph-based approximations from the same models only takes a fraction of a second. In a real development setting, one would need to stop working or shift focus while they wait for formal results, whereas the graph-based approximations can be provided almost instantly to make an informed decision. With

Table 8.2: Execution times to calculate formal and graph-based measurements for each WDS design alternative

System Design Alternative	Formal Methods Execution Time	Graph-Based Measurement Execution Time
WDS1	52.6138s	0.0060s
WDS2	41.5916s	0.0043s
WDS3	39.6907s	0.0040s
WDS4	42.3328s	0.0042s
WDS5	36.8629s	0.0041s
WDS6	37.7981s	0.0047s
WDS7	35.5755s	0.0039s
WDS8	39.8283s	0.0042s
WDS9	37.1526s	0.0040s
WDS10	39.4172s	0.0050s
WDS11	36.4172s	0.0041s
WDS12	51.7611s	0.0042s
WDS13	58.7487s	0.0042s
WDS14	41.0388s	0.0042s
WDS15	41.1811s	0.0052s
WDS16	45.6463s	0.0048s
WDS17	45.7131s	0.0062s
WDS18	42.7514s	0.0044s
WDS19	42.8829s	0.0055s
WDS20	49.1890s	0.0047s
WDS21	63.4549s	0.0064s
WDS22	57.6086s	0.0049s
WDS23	56.9613s	0.0056s
WDS24	60.4169s	0.0052s
WDS25	63.4737s	0.0070s
WDS26	63.5737	0.0049s
WDS27	63.0347s	0.0056s
WDS28	66.9186s	0.0051s
WDS29	67.2729s	0.0074s
Total Execution Time	1418s	0.1449s
Average Execution Time	48.8974s	0.0049s

a preferable design alternative selected, the formal analysis can still be performed on only the selected design alternative if needed for assurance.

Some of the discussed measures are also used to track other design concerns, such as the number of communication channels to measure coupling [SB13] or cyclomatic complexity as a complexity measure [McC76]. If a practitioner already tracks these measurements at a system or agent level, then they may already have the capability to consider the prevalence of implicit interactions without requiring additional work.

8.3 Strengths of Identifying System Properties and Design Principles

Identifying system properties, quality attributes, and design principles relevant to the mitigation of implicit interactions provides yet another avenue for considering this type of vulnerability at the design phase of a project. Any combination of formal methods, graph-based measurements, and design principles can be used depending on the practitioners' skill set to approach the threat posed by implicit interactions. A model-driven development may focus more on formal methods, while data-driven development can utilize graph-based measurements and other development cycles that do not utilize formal or semi-formal methods can evaluate and still improve a design's adherence to identified design principles to improve a system design with respect to the prevalence of implicit interactions.

Several of the identified design principles also have overlapping concerns. Where appropriate in Section 7.2 we have emphasized similarities between design principles (e.g., Cohesion and Single-Responsibility), but it is important to include each as not all designers may be well versed in every design principle. Depending on their work, different practitioners may focus on different design principles. For example, Single-Responsibility comes from the SOLID principles of object-oriented development [Mar00, MNK03] which may not be known to practitioners if they do not operate in that development paradigm. The same can be said for security design principles in how principles like Isolation and Modularity relate to Low Coupling and High Cohesion, and how Least Common Mechanism and Open-Closed Principle have overlapping concerns. Not all system designers have security experience, so identifying more widely known principles like Low Coupling and High Cohesion is useful so results are understood by a wider range of practitioners.

8.4 Limitations of the Proposed Approach and Obtained Results

A limitation common to all of the formal methods, graph-based measurements, and design principle adherence approaches is that none will tell a practitioner exactly what design changes to make. All methods will, at best, point to a specific area of a system that should be redesigned, and it is the interpretation of analysis results that informs exactly what change should be made. Some design expertise is still required

to make design changes based on this information, and different designers may come to different conclusions of the same data produced by the analyses approaches.

Another potential limitation is that we only look at aggregate measurements for implicit interactions, with all vulnerabilities considered equally important. It is possible that a user of these methods may want to protect part of a system by eliminating any implicit interactions that influence a particular agent or agents in a system design. It would be possible to compare the contribution ratios for agents in different design alternatives to look for the lowest values for a particular agent or agents, but that would only allow a designer to select the alternative with fewest implicit interactions passing though those agents, not the design alternative with the least ability to influence those agents. Analyzing the adherence to the identified design principles in a specific part of the system, rather than the system as a whole, may also help to some extent.

Specific implicit interactions with a higher exploitability, a concept discussed in Section 2.1, are also not considered when comparing design alternatives with aggregate measurements. By aiming for the fewest number of vulnerabilities, it is possible that we leave behind a small number of very exploitable implicit interactions that can still cause issues in the system once implemented. At the time of writing, it is infeasible to use the existing tool support [Jas20a] on the scale required to incorporate exploitability analyses throughout this work. Future works with more efficient methods could allow for a comparison between methods that target the most exploitable vulnerabilities for redesign with our method of minimizing the prevalence of existing vulnerabilities. Meanwhile, reducing the prevalence of implicit interactions with the

methods we discuss in this work can enable practitioners to perform an exploitability analysis on the final design alternative they obtain to understand if those vulnerabilities that remain are of much concern.

Finally, despite providing semi-formal and non-formal analysis approaches in this work, if a system requires a high level of security assurance, the formal analysis may need to be completed anyway. Some benefit is still provided by having graph-based measurements and design principles however, since the formal analysis would only need to be completed at the end of the design phase. Intermediate design iterations could be completed more quickly with the less formal approximations.

8.5 Threats to Validity

In this section, we discuss threats to validity and how each affects the strength of the conclusions of this work. Threats to validity are classified into conclusion validity, internal validity, construct validity, and external validity [CbO17].

8.5.1 External Validity

External validity is the degree to which the results generalize [CbO17]. This work employs a common practice in systems engineering and computer science related research, in which we present a methodology, show its application to one or a small set of example models, and present conclusions. Of course, identifying the relationships in Section 6.2 and design principles in Section 7.2 on our illustrative examples does not

necessarily mean all systems everywhere will always provide these same conclusions. With the agent-level graph-based approximations identified with the WDS, we have also seen a strong relationship between the agent-level implicit interaction and graph-based measurements in two other system models of a maritime port terminal and batch chemical reactor [NJ21].

Though in many instances we see strong relationships between implicit interactions and system-level measurements in both the MCCS and WDS, when we see a relationship that exists in one illustrative example but not the other, we attribute this to the differences between the two models, but it could be the case that the relationship is specific to the illustrative example and not the type of system which that example represents. For example, we only see a strong positive relationship between the number of implicit interactions and number of intended interactions in the MCCS and we conclude this relationship only exists in systems with a single control flow, but it could be the case that other systems with a single control flow do not exhibit this relationship. More work would be needed to verify or deny this conclusion for system-level measurements.

8.5.2 Conclusion Validity

Conclusion validity is the ability to derive conclusions from the relationships between independent and dependent variables [CbO17]. It is obviously not feasible to look at every possible design change for either system, but of all possible alternative designs, 10 for the MCCS and 29 for the WDS, is a fairly small sample of that total design space. The use of critical values helps to combat the small sample size, since despite

the variability we see in the plots in Section 6.2, we can still say these relationships are statistically significant with a 95% confidence level.

8.5.3 Internal Validity

Internal validity is the degree to which changes to a dependent variable exclusively caused by changes in one or more known independent variable(s) [CbO17]. We see in Section 6.2 that none of the correlation values are perfectly 1.0 and the plots do not show a perfect monotonic relationship across design alternatives. Some variability is to be expected as any one measurement is unlikely to be the only thing that impacts how many implicit interactions a system has. We simply find measurements that correlate with the prevalence of implicit interactions, and at best are just one of possibly many factors that influence the prevalence of implicit interactions in a system. There may be other factors that impact the prevalence of implicit interactions that we do not consider, as evidenced by our inability to fully explain all of the variation in measurements of implicit interactions in Section 6.2.3.

8.5.4 Construct Validity

Construct validity is the belief that the defined independent and dependent variables are a good representation of the theoretical concepts under study [CbO17]. In order for us to draw conclusions about what properties and principles are relevant to reducing the prevalence of implicit interactions in system designs, we assume that the measurements we relate to those properties and principles accurately represent

what they aim to measure. It can be quite difficult to find effective quantitative measurements for qualities, which is why we focus on existing published works that identify measurements in Chapter 7 instead of trying to come up with justifications for why a measurement relates to a specific software quality, system property, or design principle on our own.

We must also assume that the relationships between formal and graph-based measurements, and the relationships between graph-based measurements and design properties, principles, etc., are transitive. For example, we say that if the number of implicit interactions relates to the number of direct interactions, and the number of direct interactions relates to coupling, then the number of implicit interactions is, at least in part, a coupling issue. This assumption may not hold in all circumstances, or in some instances the relationship between implicit interactions and the concerns discussed in Chapter 7 may be weaker than expected by requiring several steps to jump from formal to informal analysis.

In addition to requiring the measurements to accurately capture their associated qualitative properties, the measurements must also be taken correctly. This means the tools we use to generate our data for use in Chapter 6 must be free from any faults that would impact the results they produce. This includes the Python package NetworkX [HSS08] we use for graph-based analysis, existing tool support for implicit interactions analysis [JKZ14, Jas15], and our own scripts to collect and organize measurements taken from each design alternative. We have no reason to suspect any of these tools contain faults that would impact the results, but any erroneous results they report could impact our conclusions.

8.6 Conclusions

The purpose of this chapter was to discuss the strengths and limitations of the methods used and results obtained in this work. The main takeaways from this chapter are:

1. This work does not completely eliminate the threat posed by implicit interactions, but provides several benefits and advancements to the state-of-the-art in design-phase implicit interaction mitigation.
2. The results obtained only hold under the assumptions made, and has some noteworthy, but common, threats to validity.

In the next chapter, we offer concluding statements and a short discussion on potential future works.

Chapter 9

Concluding Remarks

In this final chapter, we summarize the contributions of the thesis and possible next steps. In Section 9.1, we summarize the results of this thesis in terms of our research questions. In Section 9.2, we briefly discuss several possible future directions for this work. In Section 9.3, we offer final closing remarks.

9.1 Outcomes of the Work

In this thesis we presented several approaches to improve system security by eliminating implicit interactions at the design phase of a project. We can now revisit our research questions, first posed in Section 1.3.

RQ1. *How can we identity areas of a system design that should be modified to best reduce the prevalence of implicit interactions within the system?*

This is the subject of Section 5.2. We presented an approach for identifying weak points in system designs by analyzing the frequency of their components and communication channels in the system's implicit and intended interactions. The proposed approach is objective, repeatable, and like the implicit interaction analysis it builds upon [JV17b], can be used in the early stages of development when changes to identified weak points can be more easily made.

RQ2. *What measurable aspects of a system design are related to the prevalence of implicit interactions in that system?*

These are identified in Section 6.2. For the system level, we identified the number of direct interactions, cyclomatic complexity, and the number of cycles in the interaction model of a system design as measurements that relate highly to the prevalence of implicit interactions. The number of intended interactions in a system design can also be used when the system design contains only a single control flow. For the agent level, we identify the degree centrality, fan-out, and eigenvector centrality as measurements that relate highly to the frequency a design's agents occur in its implicit interactions.

RQ3. *How do design changes that mitigate implicit interactions relate to system properties and system quality attributes?*

This is the subject of Section 7.1. Mitigating implicit interactions is found to be in line with the goals of reducing complexity and coupling in a system design. Mitigating implicit interactions can be achieved with less concurrency in a system though the relationship is not as absolute as with complexity or coupling. Based on these properties, we suggest that improving flexibility, modifiability,

reusability, maintainability, extendibility, and understandability are all in line with reducing the prevalence of implicit interactions in system designs, while improving performance or reliability will likely increase the prevalence of implicit interactions.

RQ4. *What existing design principles align with the goal of reducing the prevalence of implicit interactions in a system design?*

This is the subject of Section 7.2. We find that improving adherence to the principles of Low Coupling, High Cohesion, Law of Demeter, Single Responsibility, Isolation, Modularity, Least Common Mechanism, Open-Closed, and Economy of Mechanism all align with the goal of reducing the prevalence of implicit interactions in a system design.

9.2 Possible Future Works

The answers to our research questions, as well as the identified limitations and threats to validity in our approach, offer several opportunities to further our understanding of, and provide additional mitigation strategies for, implicit interactions. In this section, we briefly discuss possible extensions to this work based on our contributions.

Expand the sample size: Further experiments, both in terms of additional design alternatives of the MCCS and WDS, and experimentation with entirely new case study systems, can provide further support for the findings within this work and potentially expand the list of relevant system properties, quality attributes, and design principles. Exploring other example systems would allow the consideration of other

design principles not directly applicable to the illustrative examples in this work (e.g., Interface Segregation from the SOLID principles [Mar00, MNK03]). With additional case study systems, we can also better understand under which circumstances different measurements provide better approximations, and in which circumstances different design principles are more applicable.

Implement tool support: Tool support for the extended formal analysis and graph-based approximations would greatly aid in their adoption. Existing implicit interaction tool support [JV17b] could be extended to incorporate the calculation of weak points. An application could also be created to build an interaction model in a graphical interface, to build an interaction model from C^2KA specification files like we do in our approach, or to build an interaction model from other more commonly used formats (e.g., SysML [FMS14]). Such a tool could then automate the system and agent level measurements to provide data for practitioners to make informed decisions. This could potentially be integrated with other existing automated analyses (e.g., as part of the Compass toolkit [SJY21b]) to provide a suite of security-focused design tools to enable effective security-by-design.

Demonstrate an iterative design process using the findings from this work: With another example system, we could perform a validation experiment to see how well our findings generalize by executing the "System Design Development" activities of Figure 3.2. In this hypothetical experiment, we would start with a new example system and perform several rounds of design changes based on increased adherence to the design principles we identify to answer research question **RQ4**. We would then perform a formal analysis of each alternate design only after all design changes

are completed to see how the prevalence of implicit interactions changes as a way to show that the recommendations of this work are applicable to more than just the illustrative examples we use to generate our findings.

Explore the relationship with non-interference: In section 5.3.2, we saw the extent to which the prevalence of implicit interactions in the WDS was caused by the fact that there are two separate control flow processes in the system. It would be interesting to explore this further with additional case study systems to understand how implicit interactions relate to non-interference [GM82], similar concepts such as feature interaction [WE05], and whether strategies to avoid interference or feature interaction are effective at reducing the prevalence of implicit interactions in system designs.

Identify design patterns to mitigate implicit interactions: Design patterns offer more concrete solutions to solving a design problem as they would provide more exactly the change that should be made to reduce the prevalence of implicit interactions in a system design. Several approaches to analyze and catalogue security patterns have been developed over the years [YB97, DSS+09, FB13, HW18], but it is currently unknown how these patterns relate to the notion of implicit interactions. Understanding which design principles relate to the prevalence of implicit interactions and understanding the role complexity plays can help to narrow down candidate patterns that might be applicable to mitigate implicit interactions, as well as looking for common changes between the MCCS, WDS, and other systems that all successfully reduce the prevalence of implicit interactions.

Provide mitigation methods to focus on specific vulnerabilities: If we are able to capture the potential impact an implicit interaction might have, or how likely the potential communication pathway might be exploited, then it may be possible to provide an understanding of what aspects of a design contribute to the "worst" implicit interactions, and how those specific vulnerabilities can best be targeted for remediation. The exploitability of an implicit interaction is one possible area of exploration for this direction of research, but it is infeasible to use the existing tool support for calculating this property on on the scale required [Jas20a].

9.3 Closing Remarks

Security is often discussed in terms of countermeasures and controls aimed at eliminating or mitigating the impact of malicious activity, but what we demonstrate in this work is how important the underlying design of the system behind is for limiting the reach of a compromised agent should those countermeasures fail. System designers should consider implicit interactions as a design-phase concern through one or more of the strategies discussed in this work as the design is much harder to change once implemented, and this early focus can help to avoid major issues later in the development lifecycle should an implicit interaction be exploited. By effectively mitigating the prevalence of implicit interactions in a system design, we not only make changes in line with many other design goals, but also help to make the systems we rely upon more foundationally secure.

Appendix A

Graph-Based Measurements from Illustrative Example Design Alternatives

The tables contained in this Appendix contain the node centrality values for each design alternative of the MCCS and WDS. These values are used in Chapter 6 to create Table 6.7 and Table 6.9.

A.1 Measurements taken from the MCCS Design Alternatives

Table A.1: Degree centrality for the interaction model of each alternative system design of the MCCS

System Design Alternative	C	S	H	P
MCCS1	9	6	7	8
MCCS2	9	4	6	7
MCCS3	11	4	7	8
MCCS4	12	4	6	8
MCCS5	18	4	5	9
MCCS6	14	4	4	6
MCCS7	-	5	6	5
MCCS8	-	3	5	4
MCCS9	18	4	5	9
MCCS10	14	4	4	6

Table A.2: Fan-In for the interaction model of each alternative system design of the MCCS

System Design Alternative	C	S	H	P
MCCS1	4	2	3	6
MCCS2	4	2	2	5
MCCS3	4	2	3	6
MCCS4	4	2	3	6
MCCS5	7	2	3	6
MCCS6	7	2	2	3
MCCS7	-	1	3	4
MCCS8	-	1	2	3
MCCS9	7	2	3	6
MCCS10	7	2	2	3

Table A.3: Fan-Out for the interaction model of each alternative system design of the MCCS

System Design Alternative	C	S	H	P
MCCS1	5	4	4	2
MCCS2	5	2	4	2
MCCS3	7	2	4	2
MCCS4	8	2	3	2
MCCS5	11	2	2	3
MCCS6	7	2	2	3
MCCS7	-	4	3	1
MCCS8	-	2	3	1
MCCS9	11	2	2	3
MCCS10	7	2	2	3

Table A.4: Eigenvector centrality for the interaction model of each alternative system design of the MCCS

System Design Alternative	C	S	H	P
MCCS1	0.5	0.5	0.5	0.5
MCCS2	0.5573	0.4352	0.5573	0.4352
MCCS3	0.5573	0.4352	0.5573	0.4352
MCCS4	0.5573	0.4352	0.5573	0.4352
MCCS5	0.7071	0.4082	0.4082	0.4082
MCCS6	0.7071	0.4082	0.4082	0.4082
MCCS7	-	0.5774	0.5774	0.5774
MCCS8	-	0.5774	0.5774	0.5774
MCCS9	0.7071	0.4082	0.4082	0.4082
MCCS10	0.7071	0.4082	0.4082	0.4082

Table A.5: PageRank centrality for the interaction model of each alternative system design of the MCCS

System Design Alternative	C	S	H	P
MCCS1	0.2869	0.1399	0.2524	0.3208
MCCS2	0.3336	0.1415	0.2227	0.3022
MCCS3	0.3266	0.1291	0.2443	0.3
MCCS4	0.3432	0.1396	0.2317	0.2855
MCCS5	0.4797	0.1116	0.1487	0.2599
MCCS6	0.4797	0.154	0.154	0.2123
MCCS7	-	0.1764	0.446	0.3777
MCCS8	-	0.1764	0.446	0.3777
MCCS9	0.4797	0.1116	0.1487	0.2599
MCCS10	0.4797	0.154	0.154	0.2123

A.2 Measurements taken from the WDS Design Alternatives

Table A.6: Degree centrality for the interaction model of each alternative system design of the WDS

System Design Alternative	PLC	CFP1	CFP2	SO3	SFM	SAP	OP	FAULTMNGR
WDS1	10	1	1	8	3	9	4	-
WDS2	10	1	1	6	3	6	4	-
WDS3	10	1	1	6	3	8	2	-
WDS4	10	1	1	5	3	6	2	-
WDS5	8	1	1	6	3	8	4	-
WDS6	10	1	1	9	3	10	6	-
WDS7	9	1	1	9	3	10	5	-
WDS8	10	1	1	6	3	8	5	-
WDS9	9	1	1	6	3	8	5	-
WDS10	8	1	1	5	3	6	5	-
WDS11	6	1	1	6	3	6	5	-
WDS12	7	1	1	8	3	9	4	8
WDS13	6	1	1	4	3	6	2	8
WDS14	15	1	1	9	4	8	4	-
WDS15	18	1	1	9	4	6	4	-
WDS16	15	1	1	6	4	5	2	-
WDS17	18	1	1	6	4	4	2	-
WDS18	15	1	1	9	4	8	4	-
WDS19	18	1	1	9	4	6	4	-
WDS20	15	1	1	6	4	5	2	-
WDS21	17	1	1	6	3	4	2	-
WDS22	12	1	1	9	4	8	4	8
WDS23	15	1	1	9	4	7	4	8
WDS24	12	1	1	9	4	8	4	8
WDS25	15	1	1	9	4	7	4	8
WDS26	11	1	1	6	4	4	2	8
WDS27	14	1	1	6	4	4	2	8
WDS28	11	1	1	6	4	4	2	8
WDS29	14	1	1	6	4	4	2	8

Table A.7: Fan-In for the interaction model of each alternative system design of the WDS

System Design Alternative	PLC	CFP1	CFP2	SO3	SFM	SAP	OP	FAULTMNGR
WDS1	5	1	1	5	2	3	1	-
WDS2	5	1	1	4	2	2	1	-
WDS3	5	1	1	4	2	2	1	-
WDS4	5	1	1	3	2	1	1	-
WDS5	5	1	1	4	2	2	1	-
WDS6	5	1	1	5	2	3	3	-
WDS7	5	1	1	5	2	3	2	-
WDS8	5	1	1	4	2	2	2	-
WDS9	5	1	1	4	2	2	2	-
WDS10	5	1	1	3	2	1	2	-
WDS11	5	1	1	3	2	1	2	-
WDS12	5	1	1	5	2	3	1	3
WDS13	4	1	1	3	2	1	1	3
WDS14	8	1	1	5	2	3	1	-
WDS15	9	1	1	5	2	3	1	-
WDS16	8	1	1	3	2	1	1	-
WDS17	9	1	1	3	2	1	1	-
WDS18	8	1	1	5	2	3	1	-
WDS19	9	1	1	5	2	3	1	-
WDS20	8	1	1	3	2	1	1	-
WDS21	9	1	1	3	1	1	1	-
WDS22	8	1	1	5	2	3	1	3
WDS23	9	1	1	5	2	3	1	3
WDS24	8	1	1	5	2	3	1	3
WDS25	9	1	1	5	2	3	1	3
WDS26	7	1	1	3	2	1	1	3
WDS27	8	1	1	3	2	1	1	3
WDS28	7	1	1	3	2	1	1	3
WDS29	8	1	1	3	2	1	1	3

Table A.8: Fan-Out for the interaction model of each alternative system design of the WDS

System Design Alternative	PLC	CFP1	CFP2	SO3	SFM	SAP	OP	FAULTMNGR
WDS1	5	0	0	3	1	6	3	-
WDS2	5	0	0	2	1	5	3	-
WDS3	5	0	0	3	1	6	1	-
WDS4	5	0	0	2	1	5	1	-
WDS5	3	0	0	3	1	6	3	-
WDS6	5	0	0	4	1	7	3	-
WDS7	4	0	0	4	1	7	3	-
WDS8	5	0	0	2	1	6	3	-
WDS9	4	0	0	3	1	6	3	-
WDS10	3	0	0	2	1	6	3	-
WDS11	2	0	0	3	1	6	3	-
WDS12	2	0	0	3	1	6	4	5
WDS13	2	0	0	2	1	5	1	5
WDS14	7	0	0	4	2	5	3	-
WDS15	9	0	0	4	2	4	3	-
WDS16	7	0	0	3	2	4	1	-
WDS17	9	0	0	3	2	3	1	-
WDS18	7	0	0	4	2	5	3	-
WDS19	9	0	0	4	2	4	3	-
WDS20	7	0	0	3	2	4	1	-
WDS21	8	0	0	3	2	3	1	-
WDS22	4	0	0	4	2	5	4	5
WDS23	6	0	0	4	2	4	4	5
WDS24	4	0	0	4	2	5	4	5
WDS25	6	0	0	4	2	4	4	5
WDS26	4	0	0	3	2	4	1	5
WDS27	6	0	0	3	2	3	1	5
WDS28	4	0	0	3	2	4	1	5
WDS29	6	0	0	3	2	3	1	5

Table A.9: Eigenvector centrality for the interaction model of each alternative system design of the WDS

System Design Alternative	PLC	CFP1	CFP2	SO3	SFM	SAP	OP	FAULTMNGR
WDS1	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS2	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS3	0.6206	0.2108	0.2108	0.3672	0.3672	0.4603	0.2108	-
WDS4	0.6207	0.2108	0.2109	0.3672	0.3672	0.4603	0.2108	-
WDS5	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS6	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS7	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS8	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS9	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS10	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS11	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS12	0.4485	0.1145	0.1145	0.4377	0.2141	0.3900	0.4377	0.4377
WDS13	0.4995	0.1616	0.1616	0.4462	0.2894	0.3949	0.1568	0.4846
WDS14	0.5467	0.1567	0.1567	0.4114	0.2934	0.4768	0.4114	-
WDS15	0.5651	0.1727	0.1727	0.4439	0.1727	0.4439	0.4439	-
WDS16	0.6206	0.2108	0.2108	0.3671	0.3671	0.4603	0.2108	-
WDS17	0.7071	0.2887	0.2887	0.2887	0.2887	0.2887	0.2887	-
WDS18	0.5467	0.1567	0.1567	0.4114	0.2935	0.4768	0.4114	-
WDS19	0.5651	0.1727	0.1727	0.4439	0.1727	0.4439	0.4439	-
WDS20	0.6206	0.2108	0.2108	0.3672	0.3672	0.4603	0.2108	-
WDS21	0.7071	0.2887	0.2887	0.2887	0.2887	0.2887	0.2887	-
WDS22	0.4880	0.1133	0.1133	0.4045	0.2167	0.4453	0.4045	0.4045
WDS23	0.4918	0.1182	0.1182	0.4232	0.1182	0.4232	0.4232	0.4232
WDS24	0.4880	0.1133	0.1133	0.4045	0.2167	0.4453	0.4045	0.4045
WDS25	0.4918	0.1182	0.1182	0.4232	0.1182	0.4232	0.4232	0.4232
WDS26	0.5353	0.1513	0.1513	0.4061	0.2837	0.4687	0.1224	0.4331
WDS27	0.5973	0.1978	0.1978	0.3601	0.1978	0.3601	0.1623	0.4901
WDS28	0.5353	0.1513	0.1513	0.4061	0.2837	0.4687	0.1224	0.4331
WDS29	0.5973	0.1978	0.1978	0.3601	0.1978	0.3601	0.1623	0.4901

Table A.10: PageRank centrality for the interaction model of each alternative system design of the WDS

System Design Alternative	PLC	CFP1	CFP2	SO3	SFM	SAP	OP	FAULTMNGR
WDS1	0.2779	0.0907	0.0907	0.1886	0.0916	0.1698	0.0907	-
WDS2	0.3157	0.0992	0.0992	0.1706	0.0888	0.1273	0.0992	-
WDS3	0.3136	0.0987	0.0987	0.1599	0.0862	0.1440	0.0987	-
WDS4	0.3481	0.1065	0.1065	0.1426	0.0835	0.1065	0.1065	-
WDS5	0.2619	0.1263	0.1263	0.1424	0.0884	0.1282	0.1263	-
WDS6	0.2520	0.0849	0.0849	0.1861	0.0822	0.1654	0.1445	-
WDS7	0.2504	0.0986	0.0986	0.1904	0.0865	0.1692	0.1064	-
WDS8	0.3086	0.0976	0.0976	0.1675	0.0821	0.1305	0.1161	-
WDS9	0.2728	0.1049	0.1049	0.1770	0.0856	0.1379	0.1166	-
WDS10	0.2832	0.1343	0.1343	0.1231	0.0812	0.0959	0.1478	-
WDS11	0.2503	0.1688	0.1688	0.1120	0.0889	0.0935	0.1097	-
WDS12	0.2088	0.1365	0.1365	0.1328	0.0816	0.1196	0.0676	0.1167
WDS13	0.2147	0.1365	0.1365	0.1040	0.0748	0.0777	0.0776	0.1718
WDS14	0.3222	0.0800	0.0800	0.1904	0.1043	0.1431	0.0800	-
WDS15	0.3547	0.0726	0.0726	0.1884	0.1061	0.1331	0.0726	-
WDS16	0.3802	0.0893	0.0893	0.1544	0.1083	0.0893	0.0893	-
WDS17	0.4086	0.0793	0.0793	0.1565	0.1179	0.0793	0.0793	-
WDS18	0.3222	0.0800	0.0800	0.1904	0.1043	0.1431	0.0800	-
WDS19	0.3547	0.0726	0.0726	0.1884	0.1061	0.1331	0.0726	-
WDS20	0.3802	0.0893	0.0893	0.1544	0.1083	0.0893	0.0893	-
WDS21	0.4041	0.0850	0.0850	0.1709	0.0850	0.0850	0.0850	-
WDS22	0.2619	0.0964	0.0964	0.1611	0.1140	0.1331	0.0568	0.1331
WDS23	0.2894	0.0775	0.0775	0.1699	0.1198	0.1002	0.0529	0.1039
WDS24	0.2619	0.0964	0.0964	0.1611	0.1140	0.1331	0.0568	0.1331
WDS25	0.2894	0.0775	0.0775	0.1699	0.1198	0.1002	0.0529	0.1039
WDS26	0.2767	0.0985	0.0985	0.1373	0.1122	0.0647	0.0647	0.1473
WDS27	0.3065	0.0790	0.0790	0.1471	0.1224	0.0603	0.0603	0.1455
WDS28	0.2767	0.0985	0.0985	0.1373	0.1122	0.0647	0.0647	0.1473
WDS29	0.3065	0.0790	0.0790	0.1471	0.1224	0.0603	0.0603	0.1455

Bibliography

[ACC+10] Alessandro Armando, Roberto Carbone, Luca Compagna, Keqin Li, and Giancarlo Pellegrino. Model-checking driven security testing of web-based applications. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 361–370. IEEE, 2010.

[AFC09] Bandar Alshammari, Colin Fidge, and Diane Corney. Security metrics for object-oriented class designs. In *2009 Ninth International Conference on Quality Software*, pages 11–20. IEEE, 2009.

[AFS12] Apostolos Ampatzoglou, Georgia Frantzeskou, and Ioannis Stamelos. A methodology to assess the impact of design patterns on software quality. *Information and Software Technology*, 54(4):331–346, 2012.

[AGM+19] Mehrnoosh Askarpour, Carlo Ghezzi, Dino Mandrioli, Matteo Rossi, and Christos Tsigkanos. Formal methods in designing critical cyber-physical systems. In *From software engineering to formal methods and tools, and back*, pages 110–130. Springer, Cham, 2019.

[ASKM07] KK Aggarwal, Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. Software design metrics for object-oriented software. *J. Object Technol.*, 6(1):121–138, 2007.

[BCC+20] Michel Barbeau, Frédéric Cuppens, Nora Cuppens, Romain Dagnas, and Joaquin Garcia-Alfaro. Metrics to enhance the resilience of cyber-physical systems. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1167–1172. IEEE, 2020.

[BD02] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering*, 28(1):4–17, 2002.

[Bea91] Michael Beasley. *Reliability for engineers: An introduction.* Macmillan International Higher Education, 1991.

[BHTV03] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. Modeling and validation of service-oriented architectures: Application vs. style. *ACM SIGSOFT Software Engineering Notes*, 28(5):68–77, 2003.

[BJG07] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs, Subdigraphs, Neighbours, Degrees*, page 2–6. Springer-Verlag, London, 2007.

[BNR03] Thomas Ball, Mayur Naik, and Sriram K Rajamani. From symptom to cause: localizing errors in counterexample traces. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 97–105, 2003.

[Bon87] Phillip Bonacich. A family of measures. *American Journal of Sociology*, page 1170–1182, 1987.

[BSC+19] Georgios Bakirtzis, Brandon J Simon, Aidan G Collins, Cody Harrison Fleming, and Carl R Elks. Data-driven vulnerability exploration for design phase system analysis. *IEEE Systems Journal*, 14(4):4864–4873, 2019.

[BSJ07] Koen Buyens, Riccardo Scandariato, and Wouter Joosen. Process activities supporting security principles. In *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, volume 2, pages 281–292. IEEE, 2007.

[BZ20] Srijith Balakrishnan and Zhanmin Zhang. Criticality and susceptibility indexes for resilience-based ranking and prioritization of components in interdependent infrastructure networks. *Journal of Management in Engineering*, 36(4):04020022, 2020.

[CbO17] Daniela Soares Cruzes and Lotfi ben Othmane. Threats to validity in empirical software security research. *Empirical Research for Software Security: Foundations and Experience*, 2017.

[CF14] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons, Hoboken, NJ, USA, 2014.

[CGD+16] Stephen Chong, Joshua Guttman, Anupam Datta, Andrew Myers, Benjamin Pierce, Patrick Schaumont, Tim Sherwood, and Nickolai Zeldovich. Report on the nsf workshop on formal methods for security. *arXiv preprint arXiv:1608.00678*, 2016.

[CM78] Joseph P Cavano and James A McCall. A framework for the measurement of software quality. In *Proceedings of the software quality assurance workshop on Functional and performance issues*, pages 133–139, 1978.

[Com17] Common Criteria Recognition Arrangement. Common criteria for information technology security evaluation, version 3.1 release 5, April 2017.

[CP09] Michel Cukier and Susmit Panjwani. Prioritizing vulnerability remediation by determining attacker-targeted vulnerabilities. *IEEE Security & Privacy*, 7(1):42–48, 2009.

[CTS06] Alexander Chatzigeorgiou, Nikolaos Tsantalis, and George Stephanides. Application of graph theory to oo software engineering. In *Proceedings of the 2006 international workshop on Workshop on interdisciplinary software engineering research*, pages 29–36, 2006.

[DAD+04] Menasce Daniel, Virgilio AF Almeida, Lawrence W Dowdy, Larry Dowdy, et al. *Performance by design: computer capacity planning by example*. Prentice Hall Professional, Upper Saddle River, NJ, USA, 2004.

[DBRB10] Maurice Dawson, Darrell Norman Burrell, Emad Rahim, and Stephen Brewster. Integrating software assurance into the software development life cycle (SDLC). *Journal of Information Systems Technology and Planning*, 3(6):49–53, 2010.

[Dij75] Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[DSS+09] Chad Dougherty, Kirk Sayre, Robert Seacord, David Svoboda, and Kazuya Togashi. Secure design patterns. Technical Report CMU/SEI-2009-TR-010, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2009.

[EOS09] Christian Engelmann, Hong H Ong, and Stephen L Scott. The case for modular redundancy in large-scale high performance computing systems. In *Proceedings of the 8th IASTED international conference on parallel and distributed computing and networks (PDCN)*, pages 189–194, 2009.

[FB13] Eduardo Fernandez-Buglioni. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. Wiley Publishing, first edition, 2013.

[FMM94] Marc Frappier, Stan Matwin, and Ali Mili. Software metrics for predicting maintainability. *Software Metrics Study: Tech. Memo*, 2, 1994.

[FMS14] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[Fon11] Philip WL Fong. Preventing sybil attacks by privilege attenuation: A design principle for social network systems. In *2011 IEEE Symposium on Security and Privacy*, pages 263–278. IEEE, 2011.

[Fow18] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[GCAW07] Rachana A Gupta, Mo-Yuen Chow, Avesh Kumar Agarwal, and Wenye Wang. Information security with real-time operation: Performance assessment for next generation wireless distributed networked-control-systems. In *IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society*, pages 420–425. IEEE, 2007.

[GKA17] Benjamin Green, Marina Krotofil, and Ali Abbasi. On the significance of process comprehension for conducting targeted ics attacks. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 57–67, 2017.

[GM82] Joseph A Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–11. IEEE, 1982.

[GPM04] M. Genero, M. Piatini, and E. Manso. Finding "early" indicators of UML class diagrams understandability and modifiability. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE '04.*, pages 207–216, 2004. doi:10.1109/ISESE.2004.1334908.

[Gre15] Andy Greenberg. Hackers remotely kill a jeep on the highway – with me in it, July 2015. URL: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

[GWOV08] Michael Gegick, Laurie Williams, Jason Osborne, and Mladen Vouk. Prioritizing software security fortification through code-level metrics. In *Proceedings of the 4th ACM workshop on Quality of protection*, pages 31–38, 2008.

[GZY+01] Jian Gu, Jing-li Zhou, Sheng-sheng Yu, Jie Zhang, and Zheng-cheng Duan. Study on the architecture of control system for manufacturing cells. *Journal of Shanghai University (English Edition)*, 5(2):130–135, 2001.

[Hei98] Constance Heitmeyer. On the need for practical formal methods. In *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 18–26. Springer, 1998.

[Hen01] Kennet Henningsson. Trade-offs and conflicts between quality attributes. Master's thesis, Blekinge Institute of Technology, 2001.

[HH19] Ethan Hadar and Amin Hassanzadeh. Big data analytics on cyber attack graphs for prioritizing agile security requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 330–339. IEEE, 2019.

[HIM98] Dan Hirsch, Paolo Inverardi, and Ugo Montanari. Graph grammars and constraint solving for software architecture styles. In *Proceedings of the third international workshop on Software architecture*, pages 69–72, 1998.

[HMD12] Seung Yeob Han, Karen Marais, and Daniel DeLaurentis. Evaluating system of systems resilience using interdependency analysis. In *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 1251–1256. IEEE, 2012.

[HMSW09] Charles Antony Richard Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent kleene algebra. In *International Conference on Concurrency Theory*, pages 399–414. Springer, 2009.

[HSS08] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[HW18] Brahim Hamid and Donatus Weber. Engineering secure systems: Models, patterns and empirical validation. *Computers & Security*, 77:315–348, 2018.

[Jas15] Jason Jaskolka. *On the modelling, analysis, and mitigation of distributed covert channels*. PhD thesis, McMaster University, Hamilton, ON, Canada, March 2015.

[Jas20a] Jason Jaskolka. Evaluating the exploitability of implicit interactions in distributed systems. *arXiv:2006.06045 [cs.CR]*, June 2020.

[Jas20b] Jason Jaskolka. Identifying and analyzing implicit interactions in a wastewater dechlorination system. In S. Katsikas and et al., editors, *Computer Security. CyberICPS 2020, SECPRE 2020, ADIoT 2020*, volume 12501 of *Lecture Notes in Computer Science*, pages 34–51, Cham, 2020. Springer.

[Jas20c] Jason Jaskolka. Recommendations for effective security assurance of software-dependent systems. In *Science and Information Conference*, pages 511–531. Springer, 2020.

[JK14] Jason Jaskolka and Ridha Khedri. A formulation of the potential for communication condition using C2KA. *arXiv preprint arXiv:1408.5964*, 2014.

[JKZ14] Jason Jaskolka, Ridha Khedri, and Qinglei Zhang. Endowing concurrent Kleene algebra with communication actions. In P. Höfner, P. Jipsen, W. Kahl, and M.E. Müller, editors, *Proceedings of the 14th International Conference on Relational and Algebraic Methods in Computer Science*, volume 8428 of *Lecture Notes in Computer Science*, pages 19–36, Switzerland, 2014. Springer International Publishing.

[JV17a] Jason Jaskolka and John Villasenor. An approach for identifying and analyzing implicit interactions in distributed systems. *IEEE Transactions on Reliability*, 66(2):529–546, June 2017.

[JV17b] Jason Jaskolka and John Villasenor. Identifying implicit component interactions in distributed cyber-physical systems. In *Proceedings of the 50th Hawaii International Conference on System Sciences*, HICSS-50, pages 5988–5997, January 2017.

[KB02] Hyoseob Kim and Cornelia Boldyreff. Developing software metrics applicable to UML models. *Proceedings of QAOOSE*, 2002.

[KCT+20] Luke E Kane, Jiaming James Chen, Rebecca Thomas, Vicky Liu, and Matthew McKague. Security and performance in iot: A balancing act. *IEEE Access*, 8:121969–121986, 2020.

[KKSV17] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[KM11] Matinee Kiewkanya and Pornsiri Muenchaisri. Constructing modifiability metrics by considering different relationships. *Chiang Mai Journal of Science*, 28:82–98, 2011.

[Lie89] K. J. Lienberherr. Formulations and benefits of the law of demeter. *SIGPLAN Not.*, 24(3):67–78, mar 1989.

[LM96] Daniel Le Métayer. Software architecture styles as graph grammars. In *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, pages 15–23, 1996.

[LPMH07] Juan C Lopez P, Raúl Monroy, and Dieter Hutter. On the automated correction of security protocols susceptible to a replay attack. In *European Symposium on Research in Computer Security*, pages 594–609. Springer, 2007.

[MAM+20] Sakshi Mishra, Kate Anderson, Brian Miller, Kyle Boyer, and Adam Warren. Microgrid resilience: A holistic approach for assessing threats, identifying vulnerabilities, and designing corresponding mitigation strategies. *Applied Energy*, 264:114726, 2020.

[Man03] Dino Mandrioli. The role of formal methods in developing high assurance systems: some old and some less old thoughts. In *Workshop on Software Engineering for High Assurance Systems, SEHAS*, pages 29–32. Citeseer, 2003.

[Mar00] Robert C Martin. Design principles and design patterns. *Object Mentor*, 1(34):597, 2000.

[MAS+20] Md Maruf Mortula, Md Ashraf Ahmed, Arif Mohaimin Sadri, Tarig Ali, Irtishad Ahmad, and Alaeldin Idris. Improving resiliency of water supply system in arid regions: Integrating centrality and hydraulic vulnerability. *Journal of Management in Engineering*, 36(5):05020011, 2020.

[Mat17] Lee Mathews. Criminals hacked a fish tank to steal data from a casino, July 2017. URL: https://www.forbes.com/sites/leemathews/2017/07/27/criminals-hacked-a-fish-tank-to-steal-data-from-a-casino/?sh=57a7ae1f32b9.

[MC99] Raymond E Miller and ZU Chaudry. On estimating protocol performance from a formal specification. In *1999 IEEE International Performance, Computing and Communications Conference (Cat. No. 99CH36305)*, pages 157–163. IEEE, 1999.

[McC76] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, pages 308–320, 1976.

[MFH14] Wes Maciejewski, Feng Fu, and Christoph Hauert. Evolutionary game dynamics in populations with heterogenous structures. *PLoS computational biology*, 10(4):e1003567, 2014.

[Mic] Trend Micro. Why do attackers target industrial control systems? URL: https://www.trendmicro.com/vinfo/au/security/news/cyber-attacks/why-do-attackers-target-industrial-control-systems.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall International Series in Computer Science. Prentice Hall, Upper Saddle River, NJ, USA, 1989.

[MJV17] Glenn Murray, Michael N Johnstone, and Craig Valli. The convergence of it and ot in critical infrastructure. *Proceedings of the 2017 Australian Information Security Management Conference*, 2017.

[MKPS00] S. Muthanna, K. Kontogiannis, K. Ponnambalam, and B. Stacey. A maintainability model for industrial software systems using design level metrics. In *Proceedings Seventh Working Conference on Reverse Engineering*, pages 248–256, 2000. doi:10.1109/WCRE.2000.891476.

[MKW+16] Stephen McLaughlin, Charalambos Konstantinou, Xueyang Wang, Lucas Davi, Ahmad-Reza Sadeghi, Michail Maniatakos, and Ramesh Karri. The cybersecurity landscape in industrial control systems. *Proceedings of the IEEE*, 104(5):1039–1057, 2016.

[MM20] Logan O Mailloux and Robert F Mills. Autonomous space resupply vehicle systems security design principle case study. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–8. IEEE, 2020.

[MNK03] Robert C Martin, James Newkirk, and Robert S Koss. *Agile software development: principles, patterns, and practices*, volume 2. Prentice Hall, Upper Saddle River, NJ, USA, 2003.

[MSA13] Sara Moshtari, Ashkan Sami, and Mahdi Azimi. Using complexity metrics to improve software security. *Computer Fraud & Security*, 2013(5):8–17, 2013.

[Net] NetworkX. eigenvector_centrality. URL: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.centrality.eigenvector_centrality.html#networkx.algorithms.centrality.eigenvector_centrality.

[NH19] Mark Needham and Amy E Hodler. *Centrality Algorithms*, page 77–108. O'Reilly Media, 2019.

[NJ21] Luke Newton and Jason Jaskolka. Analyzing implicit interactions to identify weak points in cyber-physical system designs. In *2021 Resilience Week (RWS)*, pages 1–8. IEEE, 2021.

[Oye15] Tosin Daniel Oyetoyan. *Dependency cycles in software systems: quality issues and opportunities for refactoring*. PhD thesis, Norweigan University of Science and Technology, April 2015.

[PC90] Andy Podgurski and Lori A. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Transactions on software Engineering*, 16(9):965–979, 1990.

[Per99] Charles Perrow. Organizing to reduce the vulnerabilities of complexity. *Journal of contingencies and crisis management*, 7(3):150–155, 1999.

[PP05] Eric Pacuit and Rohit Parikh. The logic of communication graphs. In J. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Proceedings of the 2nd International Workshop on Declarative Agent Languages and*

Technologies II, volume 3476 of *Lecture Notes in Computer Science*, pages 256–269, Berlin/Heidelberg, 2005. Springer.

[PS12] US Poornima and V Suma. Significance of quality metrics during software development process. *International Proceeding of Innovative Computing and Information Processing, ICICIP*, 2012.

[PW87] David Lorge Parnas and David M Weiss. Active design reviews: principles and practices. *Journal of Systems and Software*, 7(4):259–265, 1987.

[QFTX10] Kai Qian, Xiang Fu, Lixin Tao, and Chong-wei Xu. *Software architecture and design illuminated*. Jones and Bartlett Illuminated Series. Jones & Bartlett Learning, Sudbury, MA, USA, 2010.

[RA00] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 156–165. IEEE, 2000.

[RCF+20] Alastair Reid, Luke Church, Shaked Flur, Sarah de Haas, Maritza Johnson, and Ben Laurie. Towards making formal methods normal: meeting developers where they are. *arXiv preprint arXiv:2010.16345*, 2020.

[RD05] Octavian Paul Rotaru and Marian Dobre. Reusability metrics for software components. In *The 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.*, page 24. IEEE, 2005.

[RG12] Dev G Raheja and Louis J Gullo. *Design for reliability*. John Wiley & Sons, Hoboken, NJ, USA, 2012.

[RMO16] Ron Ross, Michael McEvilley, and Janet Oren. Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems. Technical report, National Institute of Standards and Technology, 2016.

[RS+02] CR Ramakrishnan, RC Sekar, et al. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security*, 10(1/2):189–209, 2002.

[Sah16] Amar Sahay. *Applied regression and modeling: A computer integrated approach*. Business Expert Press, 2016.

[SB13] Vinay Singh and Vandana Bhattacherjee. Identifying coupling metrics and impact on software quality. *International Journal of Engineering and Technology (IJET)*, 5(4):3433–3438, 2013.

[SB18] William Stallings and Lawrie Brown. *Computer security: Principles and practice*. Pearson Education, Upper Saddle River, NJ, USA, 4th edition, 2018.

[SJY21a] Joe Samuel, Jason Jaskolka, and George OM Yee. Analyzing structural security posture to evaluate system design decisions. In *Proceedings of the 21st IEEE International Conference on Software Quality, Reliability, and Security*, QRS 2021, pages 8–17, Hainan Island, China, December 2021.

[SJY21b] Joe Samuel, Jason Jaskolka, and George OM Yee. Leveraging external data sources to enhance secure system design. In *2021 Reconciling Data*

Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), pages 1–8. IEEE, 2021.

[Smi12] Richard E Smith. A contemporary look at saltzer and schroeder's 1975 design principles. *IEEE Security & Privacy*, 10(6):20–25, 2012.

[Sni81] Tom AB Snijders. The degree variance: an index of graph heterogeneity. *Social networks*, 3(3):163–174, 1981.

[SS75] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SW08] Yonghee Shin and Laurie Williams. An empirical model to predict security vulnerabilities using code complexity metrics. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 315–317, 2008.

[Uni] York University. Upper critical values of spearman's rank correlation coefficient. URL: https://www.york.ac.uk/depts/maths/tables/spearman.pdf.

[VGM+20] Samaikya Valluripally, Aniket Gulhane, Reshmi Mitra, Khaza Anuarul Hoque, and Prasad Calyam. Attack trees for security and privacy in social virtual reality learning environments. In *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–9. IEEE, 2020.

[WAFP19] Wojciech Wideł, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. Beyond 2014: Formal methods for attack tree–based security modeling. *ACM Computing Surveys (CSUR)*, 52(4):1–36, 2019.

[WCDM+18] Wei Wang, Antonio Cammi, Francesco Di Maio, Stefano Lorenzi, and Enrico Zio. A monte carlo-based exploration framework for identifying components vulnerable to cyber threats in nuclear power plants. *Reliability Engineering & System Safety*, 175:24–37, 2018.

[WE05] Michael Weiss and Babak Esfandiari. On feature interactions among web services. *International Journal of Web Services Research (IJWSR)*, 2(4):22–47, 2005.

[YB97] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. In *Proceedings of the 4th Annual Pattern Languages of Programming Conference*, PLoP '97, pages 1–31, Monticello, IL, USA, 1997.

[YCM20] Zejun Yang, Ying Chen, and Jose Marti. Modelling cascading failure of a CPS for topological resilience enhancement. *IET Smart Grid*, 3(2):207–215, 2020.

[YMR16] Awad Younis, Yashwant K Malaiya, and Indrajit Ray. Assessing vulnerability exploitability risk using software properties. *Software Quality Journal*, 24(1):159–202, 2016.

[YR92] Herbert P Yule and CA Riemer. Program for implementing software quality metrics. Technical report, Argonne National Lab., IL (United States). Environmental Assessment and . . . , 1992.

[ZJS11] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. A taxonomy of cyber attacks on scada systems. In *2011 International conference on internet of things and 4th international conference on cyber, physical and social computing*, pages 380–388. IEEE, 2011.

[ZN07] Thomas Zimmermann and Nachiappan Nagappan. Predicting subsystem failures using dependency graph complexities. In *The 18th IEEE International Symposium on Software Reliability (ISSRE'07)*, pages 227–236. IEEE, 2007.

[Zor94] Vasilios Zorkadis. Security versus performance requirements in data communication systems. In *European Symposium on Research in Computer Security*, pages 19–30. Springer, 1994.

[ZT06] G Zayaraz and P Thambidurai. Quantitative measurement of software architectural qualities through COSMIC FFP. In *2006 Annual IEEE India Conference*, pages 1–4. IEEE, 2006.