



2015-06-01

Integrating Synchronous Collaborative Applications with Product Lifecycle Management Workflows

Jordan Lowell Johnson
Brigham Young University - Provo

Follow this and additional works at: <https://scholarsarchive.byu.edu/etd>



Part of the [Mechanical Engineering Commons](#)

BYU ScholarsArchive Citation

Johnson, Jordan Lowell, "Integrating Synchronous Collaborative Applications with Product Lifecycle Management Workflows" (2015). *All Theses and Dissertations*. 5501.
<https://scholarsarchive.byu.edu/etd/5501>

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Integrating Synchronous Collaborative Applications
with Product Lifecycle Management Workflows

Jordan Lowell Johnson

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of
Master of Science

C. Greg Jensen, Chair
Steven E. Gorrell
Chia Chi Teng

Department of Mechanical Engineering
Brigham Young University
June 2015

Copyright © 2015 Jordan Lowell Johnson
All Rights Reserved

ABSTRACT

Integrating Synchronous Collaborative Applications with Product Lifecycle Management Workflows

Jordan Lowell Johnson
Department of Mechanical Engineering, BYU
Master of Science

Product Lifecycle Management (PLM) systems are used by thousands of engineering companies world wide. Improving these systems will have a drastic and global effect. One possible improvement is to integrate synchronous collaborative applications with PLM systems. These applications allow multiple people to work on a single digital object simultaneously. They have already been shown to reduce the time a task requires. Using these applications to complete a project will reduce the project time.

However, simply including synchronous collaborative applications within a PLM system ignores powerful benefits that could provide further time-saving benefits. The integration must allow improved awareness at the project level, so that users can mediate their own actions.

This thesis presents a method for such an integration. It also presents a prototype which implements that method. Testing was carried out using this prototype. As hypothesized, including synchronous collaborative applications shortened the overall project time. In addition, providing awareness information and allowing users to mediate themselves further shortened project times and reduced variation in those times. Proper integration should therefore provide awareness at the project level and allow users to mediate themselves to some extent.

Keywords: PLM, synchronous collaborative application, workflow management system

ACKNOWLEDGMENTS

I would not have been able to even begin this work without the guidance of Dr. Jensen. He consistently directed me in new and better directions. I also owe a debt of gratitude to my friends in the lab, who also help form my ideas into useful and helpful work. These friends include Ammon Hepworth, David French, Eric Bowman, and Jared Briggs. I also express gratitude for those who participated in any way in the testing phase of this work.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES	viii
Chapter 1 Introduction	1
1.1 Problem Statement	1
1.2 Research Objectives	3
1.2.1 Delimitations	3
1.3 Thesis Outline	4
Chapter 2 Background	5
2.1 Product Lifecycle Management Systems	5
2.1.1 PLM System History	7
2.1.2 Workflow Management Systems	8
2.2 Synchronous Collaborative Applications	9
2.2.1 Awareness	9
2.2.2 Description and Benefits	10
2.2.3 Google Docs	11
2.2.4 Synchronous Collaborative Engineering Software	12
2.3 Workflow and Synchronous Collaborative Application Integration	13
2.4 Research Purpose	14
Chapter 3 Method	17
3.1 Primary Components	17
3.1.1 Synchronous Collaborative Applications	17
3.1.2 The Workflow Management System	18
3.2 Characteristics of the Integrated System	21
3.3 Theoretical Benefits	24
3.3.1 Project Time Reduction	25
3.3.2 Continuous Collaboration	26
3.3.3 Skill-based Adaptation	26
3.3.4 Priority-based Adaptation	26
3.4 The Importance of Passive Awareness	27
3.5 Passive Awareness vs Ideal Scheduling	27
3.6 Scaling	28
3.7 Barriers to Method Implementation	28
Chapter 4 Implementation	30
4.1 Firestorm	30
4.1.1 Architecture	30
4.1.2 The Firestorm Database	31
4.1.3 The Firestorm Server	32

4.1.4	The Firestorm Client	34
4.2	User Interface	35
4.2.1	Creation Interface	36
4.2.2	Administrator Interface	36
4.2.3	Execution Interface	37
4.3	Method Compliance	40
4.3.1	Synchronous Applications	40
4.3.2	The Workflow Management System	41
4.3.3	Integrated System Characteristics	42
4.4	Firestorm Summary	42
Chapter 5	Testing and Results	43
5.1	Testing	43
5.1.1	Purpose and Approach	43
5.1.2	Environment	44
5.1.3	Procedure	44
5.2	Results	46
5.2.1	Quantitative Results	46
5.2.2	Qualitative Results	47
5.3	Discussion	50
5.3.1	Shortcomings of the Study	53
Chapter 6	Conclusions and Future Work	55
6.1	Conclusions	55
6.2	Future Work	56
6.2.1	The Effect of the Integrated System on Project Times	56
6.2.2	The use of the integrated system in more complicated projects	57
6.2.3	Handling Synchronous and Standard Applications	57
6.2.4	Modifying Check-in/Check-out	57
6.2.5	Leadership Roles in the Integrated System	58
6.2.6	Assignments in the Integrated System	58
6.2.7	Scaling Awareness	58
6.2.8	Work Monitoring	58
REFERENCES	60
Appendix A	The Firestorm Server and Client	64
A.1	Complete Source Code	64
A.2	Background	64
A.3	Firestorm: a Django Web Application	65
A.3.1	Environment	65
A.4	The Database and Server	66
A.4.1	Workflows	66
A.4.2	Workflow Logic	67
A.4.3	Workflow Creation Tests	70

A.4.4	Workflow Instances	71
A.4.5	Workflow Execution Tests	72
A.4.6	Model Event Handling	76
A.5	The Firestorm Interface	81
A.5.1	The Execution Test Interface	81
A.5.2	The FSApplication Launcher	82
A.5.3	Workflow Visualization	84
A.5.4	Possible Improvements	85

LIST OF TABLES

3.1	Breakdown of awareness information by level	24
4.1	The meaning of various colors in the workflow execution interface	40
5.1	The total project times for Workflow A by group	47

LIST OF FIGURES

2.1	A concept map for processes and activities encompassed by PLM	6
2.2	An example workflow from Teamcenter	9
2.3	A typical architecture for a synchronous application	11
2.4	The target area for the method in this research	15
2.5	An integration hierarchy	15
3.1	A theoretical workflow, with activity status clearly displayed	20
3.2	A theoretical workflow with additional awareness	22
3.3	A basic architecture for an integrated system	23
4.1	The basic architecture of Firestorm	31
4.2	A simplified ERD for workflow objects	32
4.3	The Firestorm workflow creation interface	36
4.4	A list of completed workflows	38
4.5	An example of the execution interface	39
5.1	The three variations of workflow a.	45
5.2	Workflow B (collaborative mode)	46
5.3	The total project times for Workflow A by group	47
5.4	Project times for Workflow A by test order	48
5.5	An example of the initial and adaptation phases of division of labor	49
5.6	A second example of the initial and adaptation phases	49
5.7	An example of skill-based division of labor	50
5.8	A simple engineering project	53
A.1	The User model for Firestorm	66
A.2	The Workflow model for Firestorm	67
A.3	A diagram of the object relationships defining a Workflow	68
A.4	An example workflow	68
A.5	The Workflow addAssignment method	69
A.6	Creating and authenticated Drive Service object	72
A.7	The Activity::instantiate method	73
A.8	The WorkflowExecutionModelEvent class	75
A.9	The ActivityInstance::isAccessible class	76
A.10	Opening a Firestorm document	83

CHAPTER 1. INTRODUCTION

Software is at the core of today's engineering companies. There are software applications for design, analysis, manufacturing, program management, and every task an engineer might be asked to complete. As engineering companies have grown, the need for collaboration despite geographic distance has become more and more prevalent. Again, software (in combination with the internet) has proved to be a valuable answer. Until recently, however, the same software applications that have been so instrumental in accomplishing engineering tasks have also been some of the greatest hindrances to collaboration. These applications only allow a single person to edit a file or document at a time—a distinctly non-collaborative paradigm.

This paradigm is reinforced by engineering workflows, which are often used to coordinate the efforts of an engineering team. These workflows delegate work to team members based on the assumption that only one person can work on a particular object at a time. While this ensures that all team members are well utilized, it also entrenches methods that work well only in single-user software environments.

Fortunately, recent developments in collaborative software have made it possible for multiple engineers from the same or different disciplines to work on a single digital object simultaneously, and for each to be aware of the actions of their teammates. This kind of application is called a *synchronous collaborative application*. The need to integrate these applications into commercial processes is obvious. The methods in this thesis can be used to integrate synchronous collaborative applications into an engineering workflow in a way that capitalizes on the natural human capacity to collaborate.

1.1 Problem Statement

For physical tasks, collaboration is natural and intuitive. Much information about the current situation is immediately obvious. A construction worker, for example, can easily observe

what work is currently being done, who is doing that work, what resources are available, and what progress has been made. All of this information is accurate, up-to-date, and automatic. The worker is far more likely to perform their task correctly, because they understand what their actions will mean with respect to their coworkers and the entire project.

For software-based tasks, collaboration is far less natural. Current engineering applications — such as computer-aided design (CAD), computer-aided engineering (CAE), and computer-aided manufacturing (CAM) software applications — generally limit file access to a single user. Compared to a construction worker, an engineer can observe very little about their coworkers or the state of the project. For example, as an engineer works on a CAD part or assembly, they are totally unaware of anyone else working on a separate part or sub-assembly. The single-user nature of engineering software tools prohibits the kind of collaboration inherent to physical tasks.

In order to improve collaboration, many companies have invested in product lifecycle management (PLM) systems. PLM systems help global companies coordinate the efforts of thousands of engineers on large projects throughout the entire project lifecycle. One way they do this is through a workflow management system. Workflows are commonly used by PLM systems as a method of coordination and collaboration. They usually have a visual representation, allowing all participants to quickly understand the nature of the project defined by the workflow, and how their efforts will affect the entire project. PLM systems have been adopted by most large engineering companies, in part because they provide a somewhat more effective, although not ideal, platform for collaboration.

Outside the engineering industry, many researchers have studied collaboration on digital tasks. This research has led to the creation of synchronous collaborative applications, hereafter referred to as *synchronous applications*. These applications allow multiple users to interact with a single object or document simultaneously. Changes made by one user are immediately apparent to all other users, allowing everyone to understand their actions in the context of the group. One of the most notable and commercially available examples is Google Docs. Google Docs is a word processor that allows multiple users to edit a document simultaneously, and to see all changes in real time. This kind of software can simulate the same kind of natural and immediate awareness found in physical tasks. Synchronous applications have only recently become viable, so they are not yet in common use in engineering.

PLM systems, through a workflow management system, improve collaboration at a project level, while synchronous applications improve (and often enable) collaboration at an object level. A natural step, then, is to incorporate synchronous applications with a PLM system, combining collaboration at the project level with collaboration at the object level. The potential to save time is obvious: by increasing the amount of concurrent work, projects can be completed at a faster rate.

This thesis proposes, implements, and tests a method by which synchronous applications can be integrated with PLM workflow management systems. This method enables a workflow management system to secure the benefits of collaboration at the document level as well as at the project level.

1.2 Research Objectives

There are three main objectives to this research. The first is to develop a method whereby synchronous applications can be integrated with a PLM workflow management system, and determine the benefits that such a system would have. The second objective is to create a prototype system that implements the method. The third objective is to test the prototype to see if it provides the benefits suggested by the method.

1.2.1 Delimitations

The workflow management system prototype consists of or relies on the following components:

- A custom web application based on the Django framework, including:
 - A PostgreSQL database
 - A custom workflow creation interface written in HTML and JavaScript
 - A custom workflow execution interface written in HTML and JavaScript
 - Miscellaneous administrative and training pages written in HTML and JavaScript
 - A custom client-side server written in C#
- The debug server provided by the Django framework
- A Google Drive service account
- A full installation of NXConnect, including:

- A database for storing part information
- Siemens NX
- An NXConnect server
- A modified NXConnect client

The prototype's only purpose is to demonstrate the method presented in this thesis. It contains only the minimum features required for that purpose. It is not meant for commercial use. The exact capabilities of the prototype and the purposes of each component are discussed in Chapter 4. The NXConnect server and client are described in [1].

1.3 Thesis Outline

Chapter 2 reviews some of the history and current state of PLM systems, with an emphasis on the assumptions built in to them, the nature of collaboration they provide, and their widespread adoption. This chapter also reviews related research on synchronous applications and the principles that make them powerful. Next, state-of-the-art synchronous engineering applications reviewed. Research regarding groupware integration with workflow management systems is also discussed, as this topic is closely related to the methods proposed in this thesis.

Chapter 3 presents the developed method. The final result of this method is an integrated system, where synchronous applications are successfully managed by a workflow management system. This chapter describes the components of the integrated system, its characteristics, and its benefits. This chapter also discusses the importance of the principals involved in integration, as well as some of the barriers that might exist for a implementation.

Chapter 4 presents a prototype workflow management system that incorporates several synchronous applications. The complete functionality of the prototype is discussed. The prototype's various modes of operation are also discussed. This is followed by a review of how the prototype system is used to conduct collaborative tests to determine the effectiveness of the method.

Chapter 5 presents the results of the method, which reaffirm the crucial nature of the second step of the method, workflow adaptation. Chapter 6 reviews the conclusions and future recommendation of this research.

CHAPTER 2. BACKGROUND

Integrating synchronous applications with a PLM system presents many challenges, and so it is imperative that such an integration is relevant, feasible, and beneficial. The purpose of this chapter is to establish that the method proposed in this thesis are indeed relevant, feasible, and beneficial. To this end, a discussion of three areas follows. The first section deals with PLM systems, and includes a discussion of workflow management systems. The next section introduces synchronous applications. The third section discusses the previous research dealing with integrating synchronous applications with workflow management systems. A concluding section defines the purpose of the research in this thesis.

2.1 Product Lifecycle Management Systems

In an oft-cited book [2], author John Stark gives an excellent definition of what Product Lifecycle Management entails:

Product Lifecycle Management (PLM), a new paradigm for product manufacturing, enables a company to manage its products all the way across their lifecycles in the most effective way. It helps companies get products to market faster, provide better support for their use, and manage end-of-life better. In today's highly competitive global markets, companies must meet the increasing demands of customers to rapidly and continually improve their products and services. PLM meets these needs, extending and bringing together previously separate fields such as Computer Aided Design, Product Data Management, Sustainable Development, Enterprise Resource Planning, Life Cycle Analysis and Recycling.

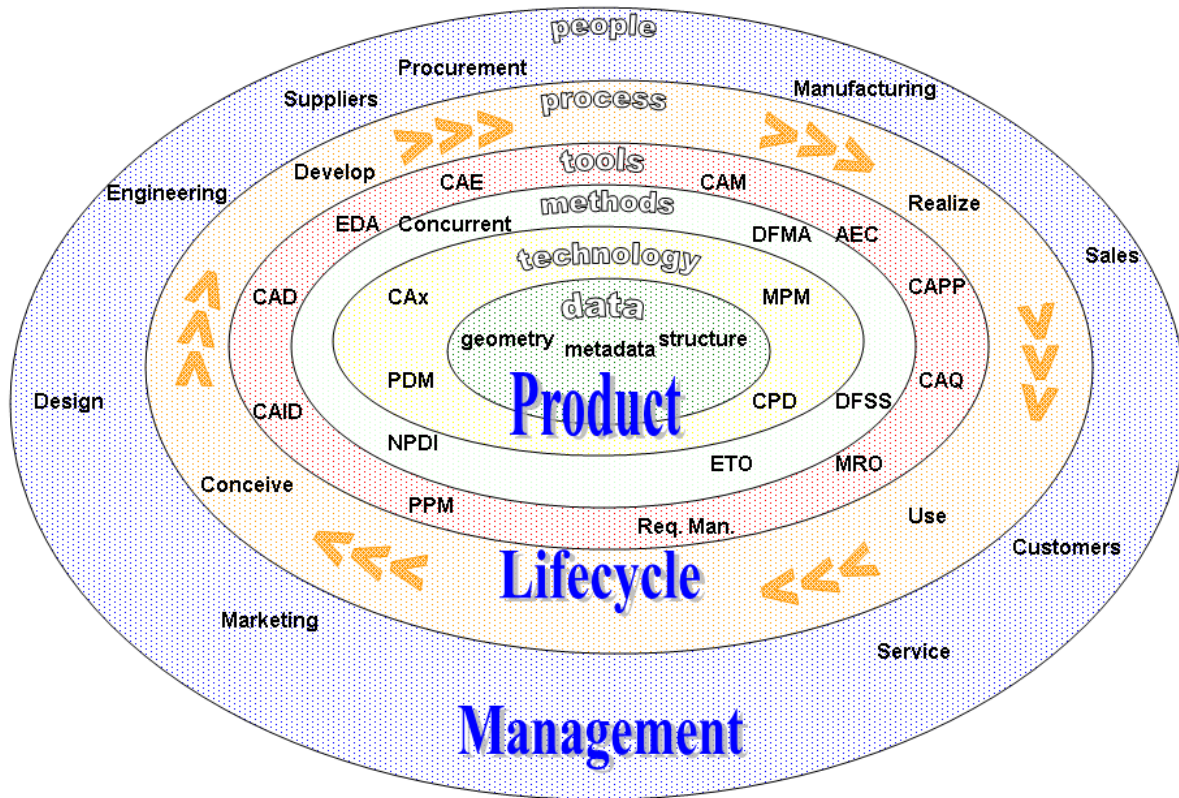


Figure 2.1: A concept map for processes and activities encompassed by PLM

PLM is a methodology that unifies all aspects of engineering and business associated with a given product. Figure 2.1¹ is very helpful in understanding PLM. It can be read from the outermost ring (people) to the center (data) in order to understand where people fit in the process and what tools they use. It can also be read clockwise to understand a product’s progression through the lifecycle. This diagram demonstrates that during a product’s lifecycle, many people (possibly thousands) will need access to product data. PLM ensures that authorized people can access relevant and up-to-date product information. However, this diagram also demonstrates that PLM is far more than a simple data repository. When PLM is implemented correctly, it is the underpinning of

¹“Product lifecycle management” by Freeformer at English Wikipedia - Own work by the original uploader. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Product_lifecycle_management.png#/media/File:Product_lifecycle_management.png

every product-related activity. Because all related processes are managed from a central system, it becomes far easier to streamline and optimize those activities.

PLM methods are becoming more and more necessary. Batenburg, Helms, and Versendaal [3] list many of the pressures and difficulties that have driven companies to adopt PLM methods:

- Competition demands shorter product lifecycles.
- Customers demand more complex products.
- Globalization demands more complex international supply chains.
- Customers demand a higher level of product customization.
- Governments demand compliance with increasing product regulations.

The same authors also list some of the benefits reported by companies who have successfully implemented PLM methods:

- Increased innovative ability
- Shorter time-to-market
- Increased profits
- Less engineering changes late in the lifecycle
- Less product faults in the field
- Higher overall efficiency

These benefits, especially in such a demanding environment, make PLM a necessity. In order to implement PLM, most companies rely on a PLM system. A PLM system is a combination of software and hardware designed to fulfill the purposes of PLM. While some companies choose proprietary solutions, there are many commercial offerings available, including Siemens Teamcenter, Dassault Systemes ENOVIA, and PTC Windchill. Because so many engineering companies have chosen to purchase a PLM system, the PLM industry is now a billion-dollar industry.

2.1.1 PLM System History

In order to understand current PLM systems, it is important to understand how they came to be. Engineers began using software tools, such as computer aided design (CAD) and computer

aided engineering (CAE) applications, as quickly as they became available. Using these applications, engineers were able to produce large quantities of data in relatively short amounts of time. These applications became so powerful that they are now indispensable to a modern engineering company.

Engineers were suddenly producing more documents than ever before, creating a need for a centralized place to store and track documents. Product Data Management (PDM) systems were created to respond to this need [4]. PDM systems (which are a combination of software and hardware) created a centralized repository for the documents produced by the engineers. An engineer could create a file (a CAD file, for example) and place it in the PDM system. Another engineer could get it, make changes, and put it back. In order to avoid duplicating and losing work, PDM systems implemented a check-out system. When an engineer checked out a CAD document, no one else could check it out until the first engineer checked it back in. In this way, no one could unknowingly duplicate work, or unknowingly erase changes made by anyone else [3, 5, 6].

Today's PLM systems evolved from the PDM systems of the past [7, 8], and have carried with them many of the assumptions made by PDM systems. One of the most fundamental assumptions carried over from legacy PDM systems is that a document or file can only be successfully edited by one individual at a time. This assumption affects many aspects of PLM systems, from file management (which is still done through a check-in, check-out system) to workflow management, discussed in Section 2.1.2. This was a good assumption because most applications operate in this fashion. However, this assumption is no longer valid due to the rise of synchronous applications, which are discussed in Section 2.2.

2.1.2 Workflow Management Systems

Workflow is defined as “The computerised facilitation or automation of a business process, in whole or part.” [9] An individual workflow represents a business process and rules by which that process will be carried out. A workflow management system is a generic software tool which allows for the creation, execution, and control of workflows [9, 10]. These systems attempt to automate many tasks, such as data transfer between computers and centralized systems. They usually provide a visual representation of the workflow, allowing creators and users to understand (1) the sequence of events that must occur and (2) the data that must be created or provided to

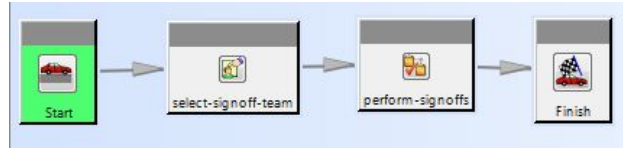


Figure 2.2: An example workflow from Teamcenter

complete a project. All of this information can be quickly understood quickly and intuitively through a visual representation.

Visual workflows have been widely adopted. In many scientific fields, visual workflows are used to help collaborating scientists understand data provenance [11–13]. Many businesses use commercial workflow management systems to gain an economical advantage [14]. It is clear that visual workflows are an extremely effective tool for collaboration and cooperation.

PLM systems provide visual workflow management tools. In fact, workflow management has been identified as one of the essential features of a PLM system [15]. By effectively utilizing workflows, a PLM system can help ensure that projects are moving as fast as they can, and that individual engineers are working effectively. In short, PLM systems use workflows to coordinate the efforts of thousands of engineers across a project. Figure 2.2 shows an example of a workflow created using Siemens Teamcenter. Although this example is extremely simple, it demonstrates how workflows can help a group of people understand the nature of the task. From this particular example, we learn that first, a sign-off team must be selected. Then, the sign-offs must be performed. Once that occurs, the task will be complete. It is easy to learn all of that information in just a glance. In the case of workflows, a picture may be worth far more than a thousand words.

2.2 Synchronous Collaborative Applications

2.2.1 Awareness

In order to understand synchronous applications (more formally known as synchronous collaborative applications), one must first understand *awareness*. Awareness is defined as an up-to-the-minute understanding of the activities of others, which provides a context for your own activity [16–18]. Awareness is best explained by physical examples. Schmidt explains awareness using an example of two people moving a table:

...by holding the table in their hands, they are both immediately aware of the state of the table: its location in space (altitude, pitch, and roll), its velocity, its weight. As soon as one of them walks slightly more briskly or slows down just a little, tilts the table to this or that side, lowers it or raises it, the changed state of the table is instantly conveyed to the other man who then has to act accordingly, by doing likewise or by counter-acting. [19] (emphasis added)

This quote is very helpful in understanding awareness. Awareness implies that multiple people have a continuous understanding of the following:

1. The state of an object
2. The current activity of others
3. The intentions of others

By understanding this information, people can effectively coordinate their activities to carry out a task. Obviously, awareness is less important if only one person can change the state of the object. However, in collaborative situations, awareness is critical. Dourish and Bellotti, in one of the earliest papers on awareness [16], write the following:

“Awareness”, principally of participants’ activities with respect to a collaborative context, is a critical issue for collaborative systems... It is fundamental to coordination of activities and sharing of information, which, in turn, are critical to successful collaboration.

2.2.2 Description and Benefits

For physical tasks, awareness is easily achieved; people can see, hear, and feel the object they wish to understand and change. For digital tasks, awareness must be intentionally provided. This is the purpose of synchronous applications. They provide awareness information about a single object to all users, and allow those users to change the object in real time. This paradigm fits the way people collaborate naturally, just as they would for a physical task [20].

Figure 2.3 shows a typical physical architecture for a synchronous application. Clients send and receive updates through an application server, which often uses a database to store user events

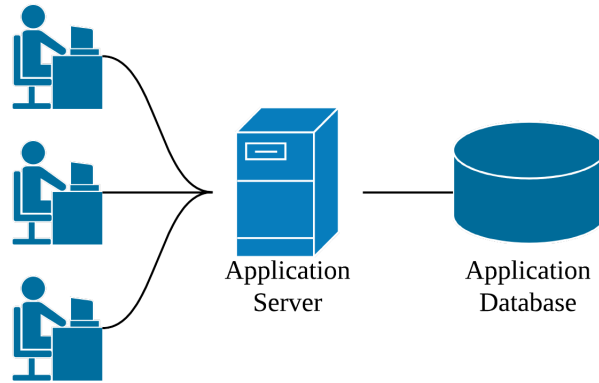


Figure 2.3: A typical architecture for a synchronous application

and document data. The server ensures that all clients are continuously synchronized. Each client provides a view of the document, as well as some awareness information, to the user.

Dourish and Bellotti [16] described the *shared feedback* approach to providing awareness information, which is the approach used by most synchronous applications. In this approach, awareness information is passively collected and distributed in the same workspace as the object of collaboration. In architecture terms, each client silently and automatically collects changes made by the user and sends them to the server for distribution. This provides users with awareness information without increasing their workload, and it creates an environment in which collaboration is continuously possible. When awareness information must be explicitly requested or collected, all users see an increase in workload, and collaboration is only possible during discrete events.

Synchronous applications offer two benefits over single-user software. First of all, the time a task takes can be reduced, because synchronous work can occur on a single object [21]. The other benefit is that people can use the provided awareness information to mediate their own actions [17]. Awareness enables them to assist the group in whatever way seems best [22]. Because synchronous applications use the shared feedback approach, these benefits come at little, if any, workload cost.

2.2.3 Google Docs

Consider a synchronous application called Google Docs. Google Docs is a server-based what-you-see-is-what-you-get (WYSIWYG) word processor. Users visit a web page that displays the document in their browser, and they can edit the document in the same way as other WYSI-

WYG editors. However, another user can visit the same page from a different computer. When this occurs, both users will see another cursor in the document other than their own, with the name of the other user next to it. As either person types, the characters generated are immediately displayed to both users, and the cursor position is updated. If a third person joins, another cursor is added to the document. Because all users are immediately aware of any changes made to the document, they can understand how to work in a way that benefits the group. This makes Google Docs an excellent example of a synchronous application, and an excellent platform to conduct further synchronous collaborative research [23–25].

2.2.4 Synchronous Collaborative Engineering Software

Software is a vital part of the modern engineering workflow. There are commercially available software packages for almost every task an engineer might be assigned. Improving engineering software inherently improves the engineering process. To this end, many researchers have sought to create synchronous engineering software.

An early example is webSPIFF, a browser-based CAD system [26]. Within webSPIFF, a server used the SPIFF CAD system and sent a view of the model to the clients. Clients could then request changes to the model. All changes were then automatically propagated to all other users. This model is often called a *centralized* collaboration model.

A more recent example is NXConnect, an extension of the NX CAD system [1]. NXConnect, like webSPIFF, uses a client-server architecture. However, NXConnect uses a *replicated* collaboration model. As clients work within NX, the NXConnect client translates their actions into commands. The server enforces command ordering and propagates the commands to all clients. Each client then applies the command. NXConnect has reached a high level of functionality, making it suitable for research on a wide variety of research topics [27–30].

Development of synchronous engineering software is not limited to CAD software. Some of this research has been aimed at finite element analysis (FEA) tools. Prototypes of such systems include CUBITConnect [31] and MUFE [32], both of which allow multiple users to synchronously prepare a finite element mesh for analysis. Preparing a complicated mesh can require weeks or even months for a single user, so allowing multiple users to aid in that process offers significant time savings.

In order to further understand the magnitude of these time savings, experiments have been carried out with both NXConnect (CAD) and MUFE (FEA). The ideal time reduction formula can be expressed as follows [33]:

$$t_n = \frac{t_1}{n} \quad (2.1)$$

In this equation, t_n represents the time to complete a task with n users, while t_1 represents the time to complete a task with a single user. In experiments, time savings have approached this limit [34]. As an example, suppose a particular part requires one week to model and six weeks to analyze, for a total of seven weeks. If two people were allowed to both model and analyze, that time could be reduced to $3\frac{1}{2}$ weeks. If three people worked on the project, the project time could be reduced to about 17 days.

Additional research in this area deals with unifying design and analysis within a synchronous collaborative setting. The end goal of this research is to allow users from multiple disciplines, such as design and analysis, to simultaneously edit a single model. This allows the time required by each discipline to overlap, further reducing the time required to complete a project [33].

Two things are obvious from this body of research. First, synchronous engineering software is becoming a reality. Prototypes like NXConnect will soon be mature enough to be considered for commercial use. Second, the potential these applications have to save time is incredible, which means that they will soon become an important part of the engineering industry.

2.3 Workflow and Synchronous Collaborative Application Integration

Because workflows and synchronous applications both center around improving collaboration, it is natural to attempt to combine them. One early attempt to do this was found in a prototype called Oz [35]. Within Oz, a workflow could be created that was composed of single-user activities. However, Oz allowed a conference activity to be part of the workflow. Once the project reached this activity, all users would participate in the conference using another synchronous application. This kind of scheduled collaboration has been repeated in the prototypes WoTel [36] and CFlow [37].

More recently, research was done on a prototype system called XCHIPS [38]. XCHIPS is a workflow management system that integrates synchronous applications and allows for unscheduled collaboration. Of all the prototypes discussed in this section, this prototype most closely follows the method outline in Chapter 3. One study used XCHIPS in a non-engineering work environment over several weeks [39]. However, the study participants used synchronous applications largely for meetings and discussions. The authors of the study report that most of the actual project work was done asynchronously.

Additionally, there is research on improving collaboration at the workflow level. Sun et al. [40] introduced a concept where multiple engineers from different disciplines could collaborate more effectively at the workflow level, in an effort to increase concurrent engineering. While this framework has the potential to increase synchronous work, it does not consider synchronous collaborative applications.

2.4 Research Purpose

Figure 2.4 compares explicit and passive awareness at the document and workflow levels. Explicit awareness implies that knowledge of the object state must be explicitly requested or generated (i.e. email or screen sharing), while passive awareness implies that knowledge of the object state is collected and distributed continuously and automatically. This research deals with passive awareness at the workflow level, or awareness of the workflow itself, rather than an individual document.

Current workflow management systems provide passive awareness at the workflow level, keeping team members aware of the status of the workflow, as well as the availability of any documents associated with the workflow. If a document is unavailable, team members are also aware of which team member currently possesses the document. However, if a synchronous application is added to the workflow, this awareness becomes inadequate. Questions concerning the availability and possession of a document become irrelevant. At the same time, questions concerning the group's involvement on a single document become extremely relevant. The awareness provided by the workflow management system must adapt to these changes, so that the benefits of synchronous applications, discussed in Section 2.2.2, can be achieved at the workflow level. These benefits are


	Document Level	Workflow Level
Explicit Awareness	Screen-sharing, email correspondence	Formal and informal meetings
Passive Awareness	Google Docs, NXConnect, etc.	

Figure 2.4: The target area for the method in this research

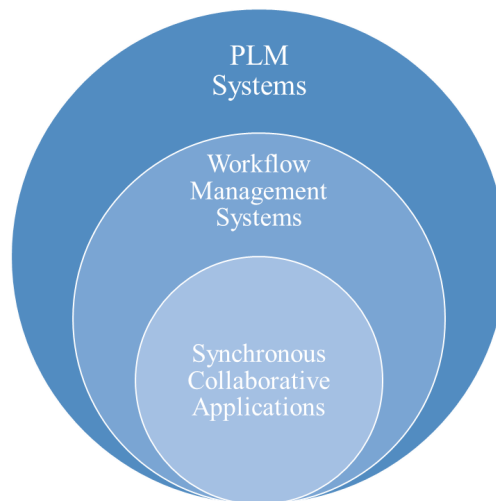


Figure 2.5: An integration hierarchy

only possible if the workflow management system uses the shared feedback approach. Without passive awareness, synchronous collaboration at any level becomes cumbersome and unhelpful.

The first objective of this research is to demonstrate a new method whereby synchronous applications can be integrated into a workflow management system. It is clear from the background research presented in this chapter that the method must have two purposes. The first is to achieve the time savings offered by synchronous collaborative applications. The second is to achieve the

benefits of passive awareness at the workflow level. Proper application of this method will allow synchronous applications to become part of PLM systems, as shown in Figure 2.5.

CHAPTER 3. METHOD

This chapter describes the method whereby synchronous applications can be integrated with a workflow management system. Properly applying this method will allow the time saving benefits of synchronous applications to be realized, while at the same time achieving the benefits of passive awareness. This chapter discusses the required components for integration, the characteristics of the integrated system, the theoretical benefits, and the possible barriers to implementation.

3.1 Primary Components

This method requires two primary components: a set of synchronous applications and a workflow management system. The requirements placed on the synchronous applications are not very strict; most current synchronous applications meet the criteria. The workflow management system, on the other hand, requires significant adaptation to be able to accommodate synchronous applications, and to provide improved passive awareness.

3.1.1 Synchronous Collaborative Applications

The first primary component required for this method is a set of synchronous collaborative applications. In order to be included in this set, an application must meet certain requirements and conform to certain assumptions. However, there is no limit to the number of applications that can be in the set, nor is there a restriction on the purpose of those applications.

The requirements imposed on the application set are not difficult to meet. They arise because an external program must be able to programmatically interact with the application. Whether this interaction occurs through an application program interface (API) or through command line options is irrelevant. The requirements are given below. The application must provide a programmatic interface that handles:

1. Creating a new document with a given name
2. Opening a document with a given name in an editing view
3. Opening a document with a given name in a read-only view

When these requirements are satisfied, it allows an external program to treat the application like a black box. This is vital because the integration method requires programmatic document creation and access. With single-user applications, automation can sometimes be achieved by manipulating files; it is trivial to copy, destroy, and sometimes even manipulate files. Synchronous applications, however, do not necessarily use files. For these applications, document manipulation can involve complicated database queries and intricate server logic. If the application does not provide an interface that handles the required operations, it may be impossible to automate those tasks.

There is also an assumption about the involved synchronous applications. It is that they provide sufficient awareness information to enable collaboration on their documents. While synchronous applications generally allow users to understand the state of the object, they might not provide as much information about the state of other users, or their intentions. However, as long as the application allows users to synchronously collaborate on a single object, it can be included in this method.

3.1.2 The Workflow Management System

The second primary component required for this method is a workflow management system. Ideally, this system would be part of a PLM system; however, the method can still be applied if this is not the case. The system required by this method is an enhanced version of the workflow management systems already in existence. Like existing systems, it must allow users to view the

workflow as a graph, and it must use the graph to provide awareness of the project status. This allows users to quickly understand many aspects of the project, such as:

- How many activities are part of the project
- Which activities are prerequisites to key activities
- Which activities are completed
- Which activities are currently in progress
- Which activities remain before project completion

A conceptual version of a workflow view is shown in Figure 3.1. In this example workflow, circles represent activities. Each activity depends on the tasks that point to it. In this workflow, for example, Task 4 depends on Tasks 2 and 3. For the purposes of this illustration, activity status is restricted to four options:

1. Not ready - not all the prerequisites have been completed for this task
2. Ready to begin - all prerequisites have been completed, but no work has begun on the task
3. In Progress - some work has been done on the task
4. Complete - all work associated with the task is complete

Even though this example workflow is simple, it demonstrates how important awareness can be. As long as this view is up to date, an observer or user can watch the progress of the project. Because this awareness information is passively collected and distributed, the user does not experience an increase in workload.

The system required by this method must also, as with existing systems, use this view to provide easy access to the documents associated with the workflow. This becomes far more important for workflows that include synchronous applications. If only one person opens the wrong document, only one person's time is wasted. If multiple people struggle to open the correct document, the amount of wasted time increases dramatically. For this reason, it is imperative that the workflow view provide easy access to the correct document.

All of the requirements discussed so far apply to workflow systems with and without synchronous applications. In order to include these applications, some additional changes must be made. One of the major changes deals with the check-in/check-out mechanism used by most existing systems. One of the purposes of this system is to prevent synchronous work on a single

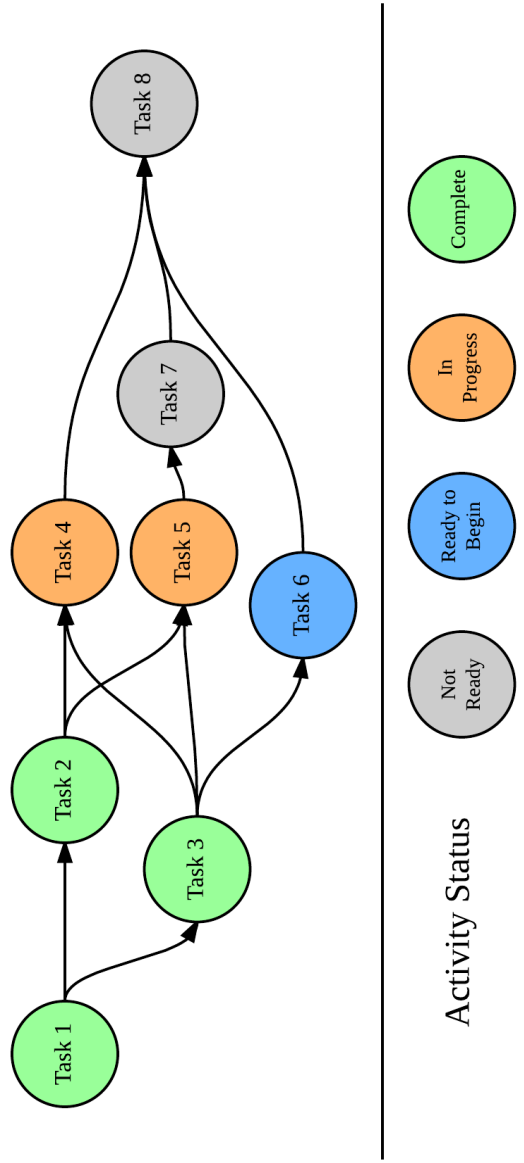


Figure 3.1: A theoretical workflow, with activity status clearly displayed

document. While this is very helpful for single-user work, it is not compatible with synchronous applications. Synchronous work must be allowed on documents that can support it. There are often other features built in to the check-in/check-out mechanism, such as an automatic revision history. Some of these features may not be compatible with synchronous applications; nevertheless, the check-in/check-out system must be modified to accommodate them. This method does not describe a way to integrate both synchronous and single-user applications; it is beyond the scope of this research.

Additionally, the workflow view must be updated so that it displays the following items:

1. a list of people who are concurrently working on a given task
2. a list of people who are investigating a given task
3. a list of people involved with the workflow, but that are otherwise idle

This is one of the more important changes because it provides new awareness information, which enables better collaboration. This new awareness information is generated by synchronous applications, and the workflow system must capture and distribute it so that it is available to the users. Figure 3.2 is identical to Figure 3.1, except that the additional awareness information is displayed. For the sake of example, assume that each task is associated with one document. Two people, John and Michael, are working on the document for Task 4, while only Robert is working on the document for Task 5. James is investigating Task 6, and Mark is idle.

The final requirement for the workflow system is that it provide workflow-level communication tools. Anyone involved with a given workflow should be able to communicate in real time with anyone else in the workflow. This can be accomplished minimally through a chat tool, but could also be extended to include voice and video conferencing. These tools must be available at all times during the project.

3.2 Characteristics of the Integrated System

The basic architecture of the integrated system is shown in Figure 3.3. It consists of a PLM server and database, which provide the workflow management system, and a set of application servers and databases. Each client is connected to the PLM server and any application servers they

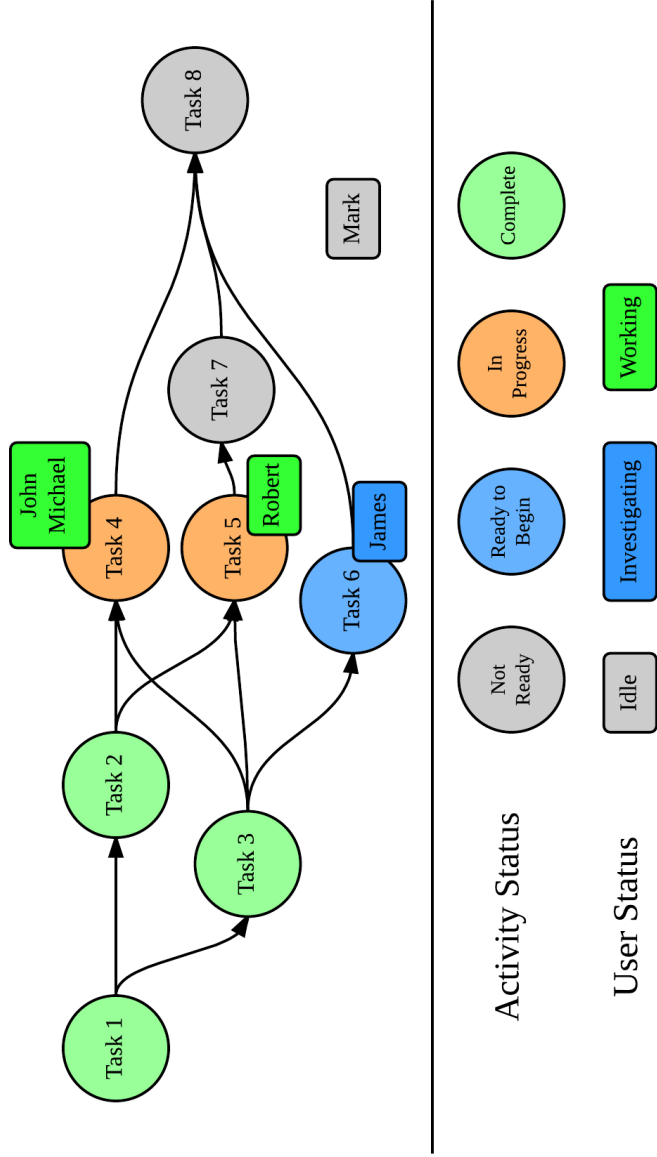


Figure 3.2: A theoretical workflow with additional awareness

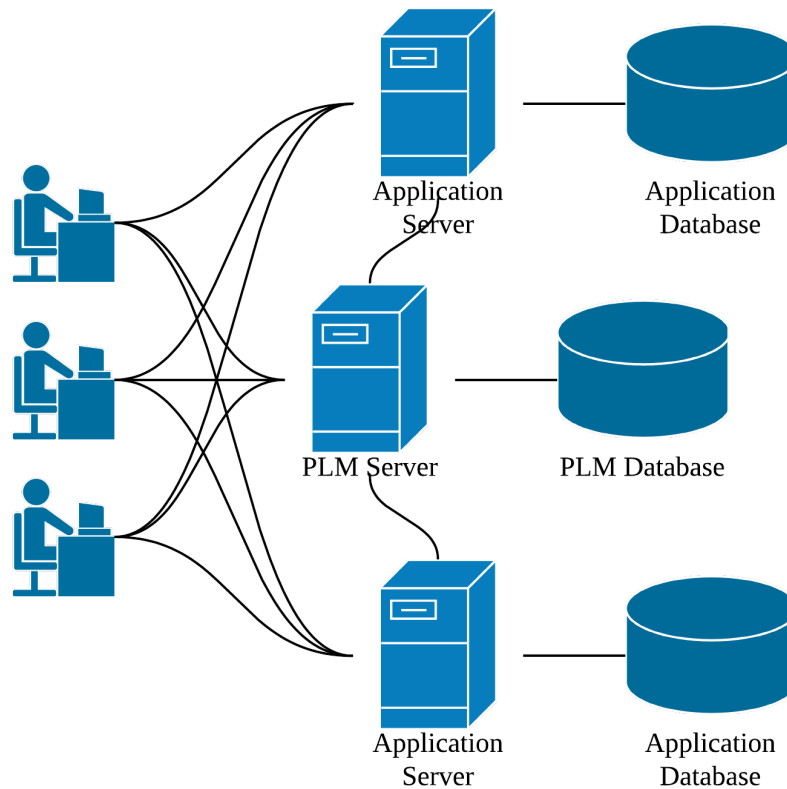


Figure 3.3: A basic architecture for an integrated system

might require. The application servers, however, are not required to have a direct link to any other application server or database.

The PLM server acts as a common link between all application servers and the clients, sending commands to create or open documents. To promote scalability, this method does not require that all network traffic associated with a specific application go through the PLM server. However, it does require that any commands to create or open a document do go through the PLM server. This allows the PLM server to track any documents that are being managed by the application servers. It also allows the PLM server to track the current working location of any client, and to provide that information to the other clients. All network traffic related to using a synchronous application should be handled by that application's server. The PLM server also acts as the communication hub for workflow-level communication, which could include chat, voice, or video conferencing.

Table 3.1: Breakdown of awareness information by level

Awareness Information	Workflow Level	Document Level
Status of the object	Activity nodes show basic completion status	Object visibly near completion (final features begin added to CAD part, etc.)
Current activity of others	Activity nodes show a list of active editors	Object is visibly changing
Intention of others	Activity nodes show a list of investigators; the workflow shows a list of others who idle but may become involved	A visual cue (cursor location, etc.) shows where a user will likely make changes
Communication	Internal tools enable workflow-level communication	Internal tools enable document-level communication

The integrated system creates a new multi-level awareness environment. The workflow management system provides awareness at the workflow level, while each synchronous application provides awareness at the document level. Table 3.1 contains examples of awareness at various levels. In general, awareness information does not travel from the document level to the workflow level or vice versa. The only exception is when a client creates or opens a document. These events affect both the document involved and the workflow, so awareness information must be collected and distributed at both levels.

3.3 Theoretical Benefits

The integrated system offers many benefits compared to current practice. The first benefit described is project time reduction. This benefit is available simply by using synchronous applications. The remaining benefits, however, are only available when an project includes synchronous applications and provides passive awareness at the workflow level. The first of these benefits is continous collaboration, meaning that collaboration can happen at any time during the project. The other benefits deal with how the group chooses to adapt to current circumstances. In some

cases, they may need to adapt to the skills required by tasks within the project. In other cases, they may need to adapt to changing task priorities.

3.3.1 Project Time Reduction

The first and most obvious benefit is an overall reduction in project time, due entirely to the use of synchronous applications. This benefit is available regardless of any awareness at the workflow level. In order to understand this benefit, suppose a project is composed entirely tasks that must be completed sequentially. In this case, the time to complete that project can be modeled as follows:

$$t_p = \sum_{i=1}^{n_t} t_i \quad (3.1)$$

Here, t_p is the time to complete the project, n_t is the number of tasks, and t_i is the time required for a single user to complete each task. Because each task in this hypothetical project is completed sequentially, the total project time is simply the sum of all the individual task times. However, if those tasks can be completed with synchronous applications, project time can then be modeled (by employing Equation 2.1) as follows:

$$t_p = \sum_{i=1}^{n_t} \frac{t_i}{n_i} \quad (3.2)$$

The new term, n_i , is the number of users assigned to each task. If only one user is assigned to every task, then Equation (3.2) reduces to Equation (3.1). It is clear from Equation (3.2) that adding more people to any task will reduce the project time. It is important to note that Equation (3.2) represents a theoretical maximum. Also, in an actual project, it is unlikely that all tasks would be completed sequentially. However, Equation (3.2) could be applied to the critical path of the project. Reducing the time of any task on the critical path of the project will reduce overall project time.

3.3.2 Continuous Collaboration

One of the most important benefits of the integrated system is continuous collaboration. While other systems and prototypes rely on pre-scheduled meetings or sessions, the system produced by this method allows collaboration at any time during the project. Because all individuals are aware of and can communicate about each other's actions in real time, they can address concerns or issues as they arise, rather than waiting for a scheduled collaboration event. Continuous collaboration enables other benefits, discussed in the following sections.

3.3.3 Skill-based Adaptation

Another benefit of the integrated system is skill-based adaptation. A project is composed of many tasks, each requiring a different skill set. Skill-based adaptation means that the group can, as they complete the project, distribute tasks to those possessing the appropriate skills. For example, those with exceptional skills in CAD modeling may gravitate towards CAD tasks. Others on the team will be aware of their decision, and without any additional communication, choose to work on another task. This is described by Sun et. al. [40]:

When collaborating synchronously, engineers can carry out the same task cooperatively in the real-time. They work intensely with one another, observing and understanding each other's intentions. Each participant contributes what they can in different fields of expertise at moments when they have the knowledge appropriate to the situation.

3.3.4 Priority-based Adaptation

The final benefit of the integrated system discussed here is priority-based adaptation. Work is often distributed based on the anticipated time it will take. Plans are made and milestones are scheduled. However, unforeseen circumstances often frustrate those plans. These circumstances can include a team member missing work, a sudden requirement change, or a task simply taking longer than anticipated. Within the integrated system, a team can easily redistribute itself on a temporary basis to handle these circumstances.

3.4 The Importance of Passive Awareness

Communication tools, such as chat, voice, and video conferencing tools, have long been used to aid in collaboration at the project level. One of the most important outcomes of successful communication is that everyone understands the state of the project. Communication has been used to establish that understanding, as well as to coordinate efforts to move forward. Communicating the state of the project is very difficult; adding synchronous applications to a project increases that difficulty. Passive awareness at the workflow level has the capacity to create that understanding *without* communication, even when synchronous applications are being used. When passive awareness is present at the workflow level, the focus of communication can move away from understanding the project and focus on more important issues.

3.5 Passive Awareness vs Ideal Scheduling

In the world of parallel computing, problem complexity can often be reduced by reducing communication between processes or threads [41,42]. Clearly, the method discussed in this chapter increases parallelization of an engineering workflow. However, it also increases the communication required between the users carrying out the work. Even though passive awareness makes that communication natural and easy, it does not remove it. In addition, each user must assess the current situation and make reasonable decisions. Again, this introduces overhead.

Ideally, each user would simply be told what to do and when to do it. Users would simply find themselves working together on various documents, thus achieving the time saving results of synchronous applications without incurring additional overhead. It is conceivable that an algorithm could be employed that directed people to work where most expedient, based on skill and priority.

This problem is known as the resource constrained project scheduling problem (RCPSP), and it is classified as NP-hard. Many algorithms exist for solving this problem [43–45], but they bring with them all the problems of heuristic approaches. They cannot guarantee that the best solution (in this case, a schedule) has been found. In addition, real-life issues might throw off the results of the algorithms. For example, if a particular person is not at work, or if any kind of delay occurs, the results from these algorithms may no longer be valid.

Using the method described in this chapter, the users solve the RCPSP, rather than an algorithm. Users, like the algorithms, are not guaranteed to find the optimal solution. However, unlike an algorithm, users can adapt the solution to current circumstances. Allowing users to solve the RCPSP will yield good and flexible results.

3.6 Scaling

There are two dimensions of scaling that might be applied to the integrated system. The first dimension is the number of people attempting to simultaneously use the system. The second is the size of the project being executed. To scale one without the other has limited use; increasing the number of people on a small project will, according to Amdahl's law [46], cease to be helpful. Conversely, increasing the project size without increasing the number of people is obviously unhelpful.

The most likely scenario is that more people will be assigned to a larger project. This creates a situation better modeled by Gustafson's law [46], which states that as long as the project size increases, speedup can be improved by adding more processors (people, in this case). As long as both dimensions are scaled together, there is no theoretical limit to either.

However, as the project scales, so does the associated awareness information. Scaling awareness information causes two issues. First of all, as the amount of activity increases, the task of collecting and propagating awareness information becomes increasingly expensive. More importantly, it will not be very easy for users to find a useful place to work; when the number of choices increases, people can be easily overwhelmed. In order for self management to be useful, awareness information must be limited to a small subdomain of a large project.

3.7 Barriers to Method Implementation

Implementing this method in a practical setting would provide significant benefits. However, there are some imposing barriers that would need to be overcome in this case. First of all, modifying the check-in/check-out feature of a PLM system to handle synchronous applications may prove to be quite difficult. It becomes even more complicated if other features, such as revision control, are built on top of the check-in/check-out system. Maintaining the features required

for single-user applications while allowing synchronous applications will not be trivial. Proper application of this method requires that features available in a PLM system still be available in the integrated system.

Once a PLM system can handle synchronous applications, another barrier presents itself. Synchronous applications are still relatively new, and few people have experience using them at all. In many cases, new methods still need to be developed for dividing work on a single document. Once these methods are developed, it will take significant training in order to move away from a single-user mindset. Projects will become less time intensive, but more personnel intensive. The cost of a process change of this magnitude can be very high.

It is the author's belief that in spite of these barriers, the benefits of the integrated system produced by this method justify its implementation.

CHAPTER 4. IMPLEMENTATION

In order to test this method, a prototype system, arbitrarily named Firestorm, has been implemented. This chapter reviews the implementation, describing its features and characteristics, and discusses its compliance with the method described in Chapter 3.

4.1 Firestorm

Firestorm is a workflow management system that integrates synchronous applications as described in Chapter 3. It incorporates NXConnect, Google Docs, and Google Sheets, and provides a workflow interface with extended passive awareness. It allows workflows to be defined, instantiated, and executed. The following sections briefly describe Firestorm at a high level; for a more detailed description, and for information on how to access the complete source code, see Appendix A (p. 64).

4.1.1 Architecture

Figure 4.1 shows the basic architecture of Firestorm. For simplicity, only one client is shown, though multiple clients can be simultaneously connected in the same way. Firestorm includes three synchronous applications: NXConnect, Google Docs, and Google Sheets (a spreadsheet editing application). Google Docs and Google Sheets are both managed through the Google Drive server and database. NXConnect is managed through the NXConnect server and database. The remainder of this section (Section 4.1.1) discusses the role of each Firestorm component.

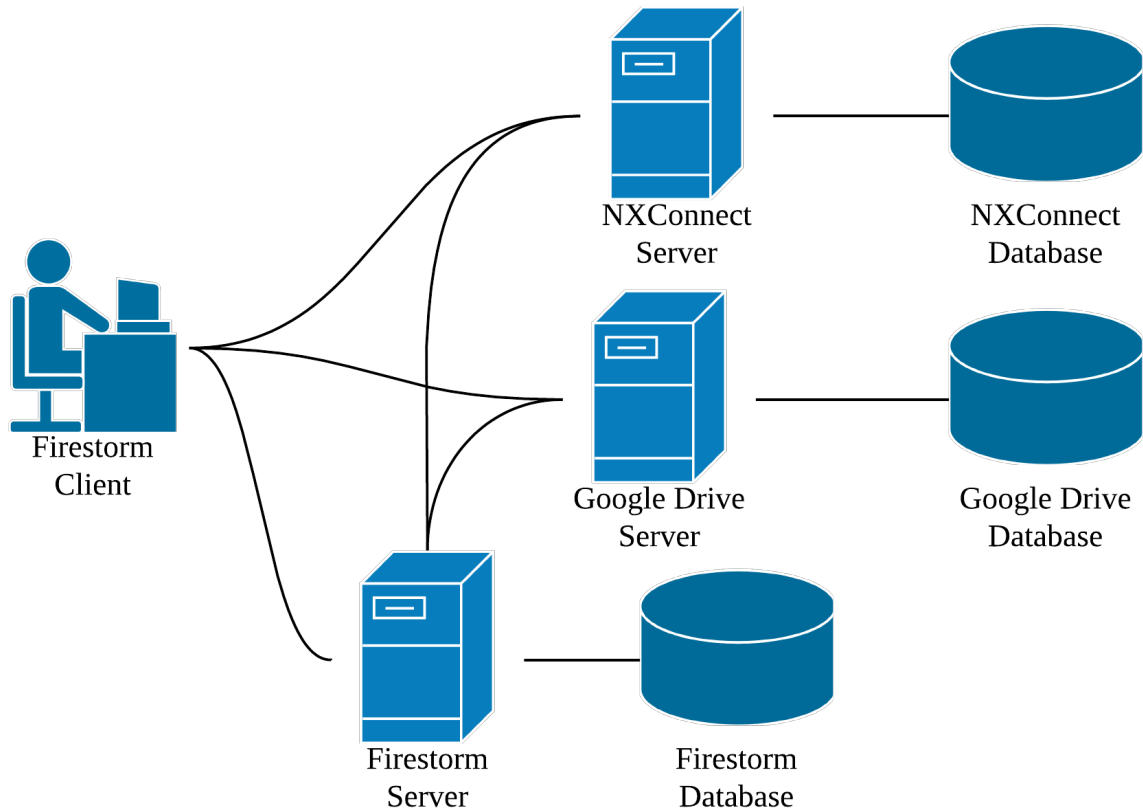


Figure 4.1: The basic architecture of Firestorm

4.1.2 The Firestorm Database

The foundation of Firestorm is a PostgreSQL database. This database stores the definition of each workflow as a collection of activities, links, and assignments. Figure 4.2 shows a simplified entity relationship diagram (ERD) for the workflow object. Each activity is assumed to be connected with one document, called an artifact. When the workflow is defined, each activity is assigned an artifact type, which can be one of the following options:

- A CAD document
- A text document
- A spreadsheet

Links are used to define the dependencies between activities. Each link points to a previous and next activity, where the next activity depends on the previous. Each assignment references a user and an assignment, meaning that the user is assigned to the activity.

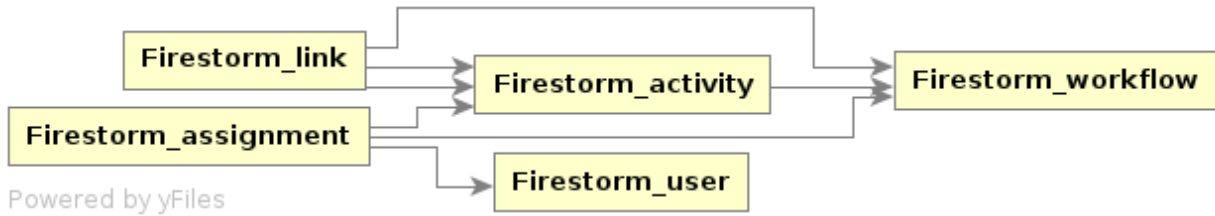


Figure 4.2: A simplified ERD for workflow objects

Once a workflow has been created, it can be used as a template to create multiple projects. Within Firestorm, this is called *instantiation*. During instantiation, the activity’s artifact type is used to create an empty document, managed by the appropriate application. The Firestorm server (Section 4.1.3) handles the creation process. The data necessary to access these documents is then stored in the database.

A workflow instance can then be *executed*, or carried out. The database represents a workflow execution as the workflow instance involved and two lists of events. The first list records the actions of a user, such as clicking or double clicking on an activity node. The second list records the results of that action, if any; these results are generated by the server according to the workflow logic. The state of the workflow at any time during execution can be recreated by reapplying the user events. A given sequence of user events is guaranteed to have the same effect on the workflow.

4.1.3 The Firestorm Server

The Firestorm server is the central component of Firestorm. Because Firestorm is primarily a web application, the server’s core responsibility is to handle user requests, and respond appropriately. Consequently, the server contains all the logic required to create, instantiate, and execute workflows. The server was written using the Django framework (a python-based web framework), which provides URL routing and an object relational manager (ORM) for databases. URL routing provides empty callback methods that allow for custom actions based on a URL. The ORM provides an object-oriented abstraction to database tables, making database manipulations very simple. In Firestorm, every action available to the user has been mapped to a URL and handled by a custom callback. Many of these require the server to access the database, which is always done through the ORM.

When a user requests a new workflow, the server creates a new workflow in the database. As the user requests changes to the workflow (such as adding and removing activities and links), the server updates the database to match the user's requests. When the user wishes to finish the workflow, the server checks the database version of the workflow to see if it complies with the set of rules for Firestorm workflows. If it passes, the workflow is marked as valid in the database.

When an administrator requests that a particular workflow be instantiated, the server first ensures that the workflow is valid. If it is, the server queries the database for a list of all activities in the workflow. For each activity, the server sends requests to create a document for the activity. If the activity is a CAD activity, the server sends a request to NXConnect to create a new part. If the activity represents a document or spreadsheet, the server sends a request to the Google Drive server to create a new document or spreadsheet. The server then stores all this information with a new workflow instance in the database.

When a user requests a view of a workflow instance, the server retrieves the workflow information from the database and sends it to the client. The server also retrieves the list of artifacts that correspond to the workflow instance, along with any related events, and sends those to the client. The client uses this information to create a view of the workflow. As the user makes changes to the workflow, the server checks each request and generates the appropriate response events, storing them in the database. During execution, the client frequently requests the latest updates to the workflow, which provides awareness of the state of the workflow, as well as the activities of others.

The server is also responsible for enforcing the mode of a workflow during creation and execution. There are three possible modes for a workflow: standard, synchronous, and collaborative. During creation, the mode affects how many people can be assigned to an activity. In standard mode, only one user can be assigned to an activity. This is meant to represent current workflow use with single-user applications. In synchronous mode, multiple people can be assigned to the activity. In collaborative mode, no one can be assigned to the activity.

During workflow execution, the server is also responsible for managing document access. Document access is always restricted based on dependencies: an activity's document is only accessible if all prerequisite activities are complete. In addition, document access can be restricted based on assignment, which is affected by the mode of the workflow. In standard mode, only the

user who has been assigned to an activity can access the document for that activity. This is also true for synchronous mode, except that multiple users may be assigned to a single activity. In collaborative mode, there are no assignment restrictions. The server is responsible for enforcing these rules.

4.1.4 The Firestorm Client

Each client computer has the following software installed:

- Google Chrome (version 41 or later)
- The NXConnect client, including an installation of NX
- The Firestorm application launcher

Google Chrome is a modern web browser. It is used as the Firestorm interface, allowing users to view and manipulate workflows. It is also used as the client for Google Docs and Google Sheets, both of which are browser-based editors. Documents can be accessed and edited by visiting a specific URL.

The NXConnect client is the client portion of NXConnect, more fully described in [1]. In order to work with Firestorm, however, a simple modification has been made. This modification allows NXConnect to accept command line options for two cases. The first case uses an NXConnect client to connect to the NXConnect database, create a part with a specified file name, and then close. This allows the Firestorm server to automatically create parts. The second case allows a specified user to log in to NXConnect and open a specified part. This allows Firestorm to provide easy access to a given NXConnect part.

The Firestorm application launcher is a client-side server. It is necessary because Firestorm provides a browser-based client, but also needs to be able to launch non-browser applications. Due to security restrictions, it is very difficult to launch external applications from within Google Chrome. The Firestorm application launcher is a simple server that connects to the Firestorm server on behalf of the client. The Firestorm server can send commands to the application launcher to launch NXConnect (using the command line parameters discussed above) or Google Chrome, directed towards a specific URL using Chrome's command line options. In order to use Firestorm, this application must be running.

The browser component of the Firestorm client is the most complicated, because all user interaction with Firestorm happens through this component. In order to create a workflow, the user visits the workflow creation page. The server returns a web page that provides a creation interface built on JavaScript. As the user edits the workflow visually, the creation interface translates those actions into requests and sends them to the server. Based on the server's response, the changes are kept or discarded.

An administrator can visit a page displaying a list of workflows. They can send a request to instantiate any valid workflow. If there are CAD activities in the workflow, then the server will send a request to the client's application launcher to launch the NXConnect client; the request contains a list of all the parts to be created. Once the NXConnect client has finished creating the parts, it closes. The client does not participate in any other part of instantiation.

A user can also visit a page that allows them to work with a team as part of a workflow execution. This page contains JavaScript that requests the workflow model from the server. The script on this page also starts an update loop, where a request is sent several times a second to the server, requesting the latest events. The script continually redraws the workflow in its latest state. This is perhaps the most important role of the client, because it is through this perpetual update that awareness is maintained. When the user requests access to a document, that request is sent to the server. If the request is valid, then the server sends a request to the client's application launcher to open the appropriate application, viewing the correct document. In this way, the client also provides easy access to the documents associated with the workflow. Once the application is launched, the appropriate application server handles requests related to viewing or changing the document.

4.2 User Interface

All interaction with Firestorm is meant to happen through a provided user interface. This includes an interface for workflow creation, instantiation, and execution. The remainder of this section (Section 4.2) describes the interfaces provided by Firestorm.

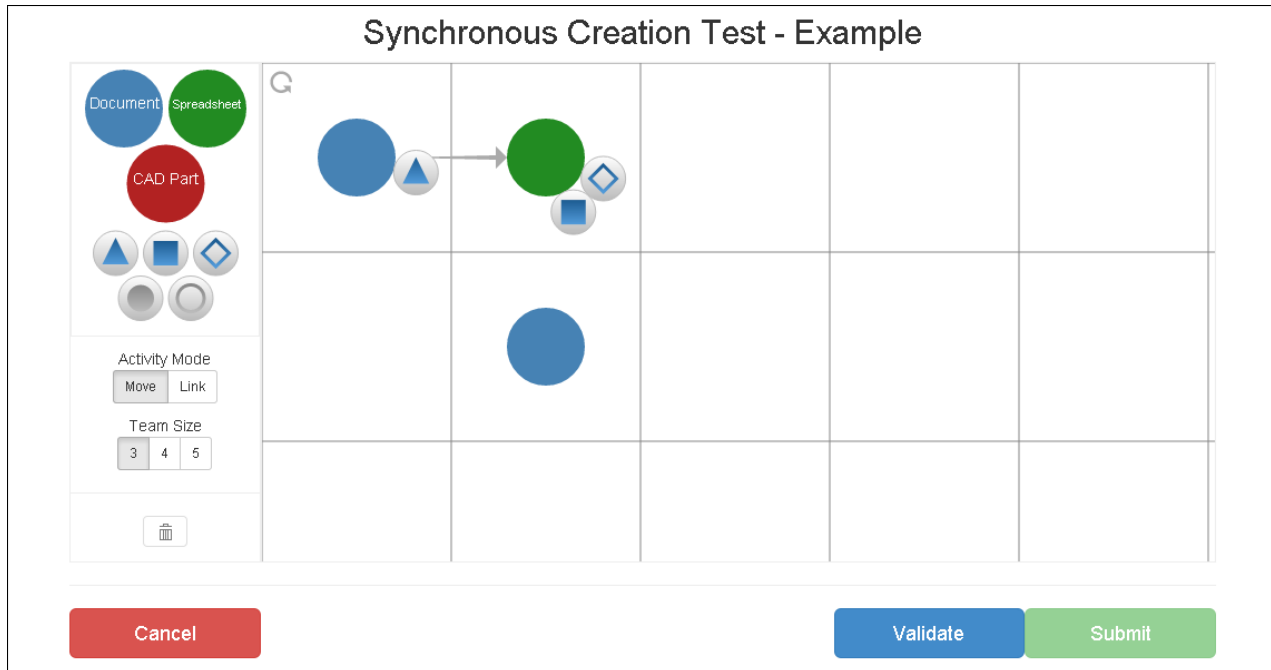


Figure 4.3: The Firestorm workflow creation interface

4.2.1 Creation Interface

Firestorm provides an easy way to create simple workflows. Figure 4.3 shows the creation interface, where a creation is in progress. The nodes represent activities; blue implies a node is a document node, green implies a spreadsheet, and red implies a CAD part. The arrow specifies a link, pointing from the previous activity to the next. The shape icons represent users. When attached to a node, the shape icons represent assignments. In this workflow, the triangle user has been assigned to a document, and the square and diamond users have been assigned to a spreadsheet. Activities, links, and assignments are all created using drag-drop logic. Once the user has completed the workflow, they must click the Validate button before they can submit the workflow.

4.2.2 Administrator Interface

Firestorm also provides an administrator interface. Figure 4.4 shows that interface. From here, valid workflows can be easily instantiated; the administrator simply has to click the “Instantiate” button. Also, this page provides a list of workflows that are currently being executed.

Using the links in this part of the list, the administrator can easily observe any current workflow execution.

4.2.3 Execution Interface

The most important interface in Firestorm is the execution interface. This is the interface that provides workflow-level awareness to all users during workflow execution. Figure 4.5 shows an example of the workflow execution interface. This interface allows three main actions: investigating an activity, opening a document, and marking a document as complete. A user can investigate an activity by left-clicking on an activity node. This updates their screen to display some relevant information about the activity. A user can also open a document by double-clicking the activity node. As long as the user has permission to open that document, the appropriate application will be launched on the client's computer, with the correct document already opened. Finally, the user can mark the document as complete. This is the only task that creates work for the user during workflow execution. When a user selects an activity, their client will update to display a list of users who still need to mark the document as complete.

For the most part, all users see the exact same workflow view during execution. As the workflow progresses, the interface updates to reflect the latest awareness information. Table 4.1 describes the meaning of each color in each context. There are a few features that change the look of the workflow on a per-user basis. The most important is activity availability; if one user is assigned to an activity and the other is not, their respective interfaces will reflect that change. Each user also has the ability to select a node by clicking. The black ring, which signifies that the node is selected, is unique to each user. When a user hovers over an activity with the mouse, it is highlighted with a grey ring. This does not affect any other user's view.

The mode of a workflow affects the appearance of the execution interface. In both standard and synchronous modes, the shape icons attached to activity nodes represent users assigned to that activity. In collaborative mode, there are no assignments. However, a shape icon may still appear next to an activity node. This indicates that a user has clicked on that node to investigate the activity. Awareness of investigation is not provided in any other mode.

Workflows	
Workflow ID	Name
90	Example <input type="button" value="Review"/>
89	Example <input type="button" value="Review"/>
88	Synchronous 3 No CAD <input type="button" value="Instantiate"/> <input type="button" value="Review"/>
87	Collaborative 3 No CAD <input type="button" value="Instantiate"/> <input type="button" value="Review"/>

Figure 4.4: A list of completed workflows

Test 82: Synchronous 3 No CAD

Name: Document 3 Part 1 **Type:** Document **Status:** In progress

Complete

The following users still need to complete this activity:

- Diamond
- Triangle (You)

Chat (Triangle)
Triangle: Square, you can start the spreadsheet now

Cancel

Figure 4.5: An example of the execution interface

Table 4.1: The meaning of various colors in the workflow execution interface

Color	Meaning
<i>Node Color</i>	
Blue	A document is associated with this activity.
Green	A spreadsheet is associated with this activity.
Red	A CAD part is associated with this activity.
Grey	The document associated with this activity is unavailable.
<i>Node Ring Color</i>	
No Ring	The activity has not been begun.
Orange	The activity is in progress.
Teal	The activity is complete.
Black	The activity is selected.
Grey	The mouse is hovering over the activity.
<i>User Icon Fill Color</i>	
Grey	The user is not currently in the workflow.
Blue	The user is in the workflow.
Green	The user is working on the document.
Purple	The user is investigating a node.

4.3 Method Compliance

The Firestorm system complies with the method specified in Chapter 3. This section reviews the requirements of the method and demonstrates that Firestorm is a satisfactory prototype.

4.3.1 Synchronous Applications

The synchronous applications in Firestorm — NXConnect, Google Docs, and Google Sheets — all comply with the requirements in Section 3.1.1. The only exception is that NXConnect does not provide a read-only access to a part through its command line interface. However, Firestorm does not provide a way to access a document for read-only access, even though Google Docs and Google Sheets both provide such a view easily. Since Firestorm doesn't manage valuable or proprietary data, this requirement can be relaxed.

4.3.2 The Workflow Management System

This section lists the requirements from Section 3.1.2 and also describes how Firestorm meets those requirements.

The system must provide awareness of the workflow through a visual graph.

Through its execution interface (see Figure 4.5), Firestorm provides users with the necessary awareness of the workflow using a graph. It passively collects and distributes awareness information. The only piece of information that is not passively collected is whether a document is complete; the user must provide that information manually. However, creating a system to automatically determine if a CAD part is finished would be extremely difficult, and so that task falls to the user, who simply has to check a box when they are finished.

The system must provide easy access to documents.

Firestorm provides very simple access to the documents managed by the workflow. Within the execution interface, a document is only a double-click away at any time. As long as the user double-clicks on the correct node, they will find themselves in the correct document.

The system must modify or remove the check-in/check-out system for synchronous applications.

Because no single-user applications are included in Firestorm, it does not have a check-in/check-out system. This is functionally the same as removing the system.

The system must provide awareness of all team members.

Firestorm uses shape icons to represent users (see Figure 4.5). In collaborative mode, these icons move to indicate their user's location. Firestorm does not display a list of idle team members, but that information is known because, within Firestorm, the team size is fixed and known to all team members.

The system must provide a workflow-level communication tool.

Firestorm provides a chat window corresponding to a workflow execution (see Figure 4.5). Only messages pertaining to that workflow execution are shared in the chat window. Any team member is allowed to use the chat tool at any time.

4.3.3 Integrated System Characteristics

Firestorm possesses the characteristics of the integrated system described in Section 3.2. From an architecture perspective, Firestorm matches the prescribed architecture. Firestorm's server is the central hub of communication, and it manages the clients and synchronous applications. Firestorm also provides the multi-level awareness described in Table 3.1. This includes a separation of awareness between the workflow and document levels. It also provides the extended passive awareness required by the method. Firestorm can easily track when activities are investigated, and when documents are opened, and it can easily distribute that information to all clients.

4.4 Firestorm Summary

Firestorm is a valid prototype of the method described in this thesis. It meets the requirements and possesses the characteristics belonging to the integrated system. It can therefore be used to test the method, to determine if the method provides the benefits it claims to provide. Firestorm was indeed tested; the execution and results of those tests are discussed in the next chapter.

CHAPTER 5. TESTING AND RESULTS

5.1 Testing

5.1.1 Purpose and Approach

The purpose of the tests carried out using Firestorm was to determine if it provides the benefits listed in Section 3.3. These are as follows:

- Time savings due to synchronous collaborative applications
- Benefits of continuous collaboration, including:
 - Skill-based adaptation
 - Priority-based adaptation

In order to determine whether or not each of these benefits was provided, it was necessary to test three kinds of workflow management systems. The first kind of workflow system does not use synchronous applications. This is comparable to current art. The second kind of workflow system does use synchronous applications, but does not improve the passive awareness at the workflow level. The third kind of workflow includes synchronous applications and provides improved awareness.

Testing each kind of workflow involves recording the task and project times, providing quantitative data. Testing also involves recording observations, providing qualitative data. By comparing the times recorded for the first and second kinds of workflows, it is possible to observe the time benefits of synchronous applications. By observing collaboration that occurs using the third kind of workflow, it is possible to determine whether or not the benefits of continuous collaboration are present.

Firestorm was designed as a testing prototype. The three workflow modes available within Firestorm reflect the three kinds of workflows that must be compared. The database representation

of events implies that each event can be easily time-stamped, providing automatic, silent, and high-precision timing during a workflow execution. Those events can also be replayed, to observe the actions of a team after the fact. This, along with recorded observations, allows for easy collection of qualitative data.

5.1.2 Environment

In order to carry out testing on the Firestorm system, a computer lab was prepared. This lab had an NXConnect server and database, a Firestorm server and database, and six client computers, each with all necessary software installed, including the NXConnect client. The Firestorm server and clients all had access to the internet, enabling access to Google Docs and Google Sheets. Essentially, a complete Firestorm system with six clients was created.

5.1.3 Procedure

In order to test Firestorm, several groups of volunteers were asked to complete several workflow projects. No volunteer participated in the study more than once. The majority of these volunteers were asked to complete a workflow called Workflow A, shown in Figure 5.1¹. This workflow was designed to be neither entirely serial nor entirely parallel; it is otherwise arbitrary.

Each task within the workflow requires the users to recreate pre-existing documents. The three text documents contain excerpts on various topics. Each text document is roughly 1500 characters in length. The first spreadsheet contains three columns of 20 normally distributed values. Users were asked to enter each value, and then find the average and standard deviation of each column. The last spreadsheet contains data from 12 college football games in 2014, including the team names, the score for each quarter, and the final score. Users were asked to enter this data and then find the average score per quarter, as well as the average final score. These tasks were selected because they require no engineering experience and very little software training, while still requiring several minutes to complete. Because the majority of the volunteers had little or no CAD modeling experience, Workflow A does not contain a CAD task.

¹The activity names for this workflow are: (first row) Document 1 Part 1, Document 3 Part 1, Document 2 Part 1 (second row) Spreadsheet 1 Part 1, Spreadsheet 2 Part 1

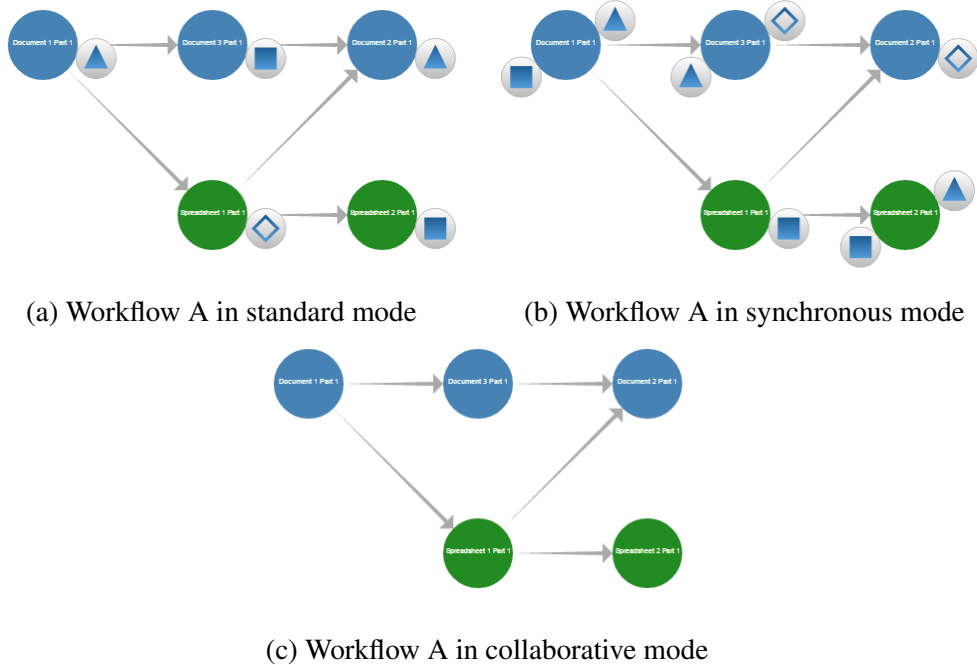


Figure 5.1: The three variations of workflow a.

Four groups, each a team of three, were asked to complete Workflow A three times, once in each workflow mode. The three workflows were completed in a random order in each group. The assignments in standard mode were arbitrary. The assignments in synchronous mode are meant to enforce reasonable collaboration, or what a team might do if otherwise free to act for themselves.

One group, a team of four, was asked to complete Workflow B (Figure 5.2, which does contain a CAD task, which consisted of creating 20 cylinders at specified locations. The structure of the workflow is identical to Workflow A, except that the second text document (chronologically) is replaced by a CAD task. In addition, the tasks were made slightly longer, so that the per-person work in each task remained approximately the same. The CAD task required the creation of 20 cylinders of any height anywhere in the part.

Volunteers were told that they would be randomly assigned to a user; they were not able to choose whether to be the triangle, square, diamond, or ring. The volunteers were unaware of the fact user roles were determined on a first-come, first-serve basis. The first person to visit the workflow execution interface became the triangle, the second became the square, the third became the diamond, and the fourth became the ring. In general, all volunteers assigned to a specific

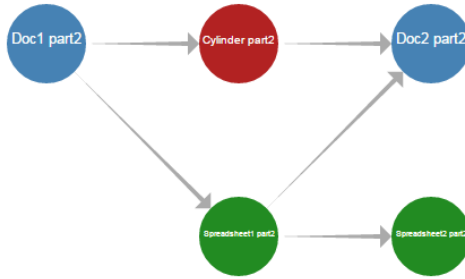


Figure 5.2: Workflow B (collaborative mode)

workflow visited that workflow’s execution page as soon as the workflow was instantiated. None of the volunteers ever attempted to acquire a certain role.

For all tests, users were collocated and allowed to converse freely. This allowed awareness beyond what was provided by Firestorm. For this reason, participants did not utilize Firestorm’s chat tool. However, participants still relied on the interface for a common context of understanding. This was reinforced by the fact that both document access and task completion were only possible through the interface.

During each test, an administrator was present to answer questions about the tasks, and to record observations. When each group had completed its set of tests, test participants were asked about their experience using Firestorm. These observations and responses both provided valuable qualitative data.

5.2 Results

5.2.1 Quantitative Results

The total project time was measured in seconds as the time from when the first user came online to when the last activity was completed. The total project time required to complete Workflow A in the three workflow modes is presented in Table 5.1, along with the average time and standard deviation for each collaboration mode. The total project times are also represented graphically in Figure 5.3. Group 2 was unable to complete the standard mode workflow. The total project time for Workflow B was 914.7 seconds.

Table 5.1: The total project times for Workflow A by group

Group	<i>Workflow Mode</i>		
	Standard	Synchronous	Collaborative
Group 1	1076.2	1222.3	843.4
Group 2		1255.9	862.2
Group 3	1800.5	1060.2	923.9
Group 4	1512.7	1292.9	934.3
Average	1463.1	1207.8	890.6
Std. Dev.	354.7	102.6	44.9

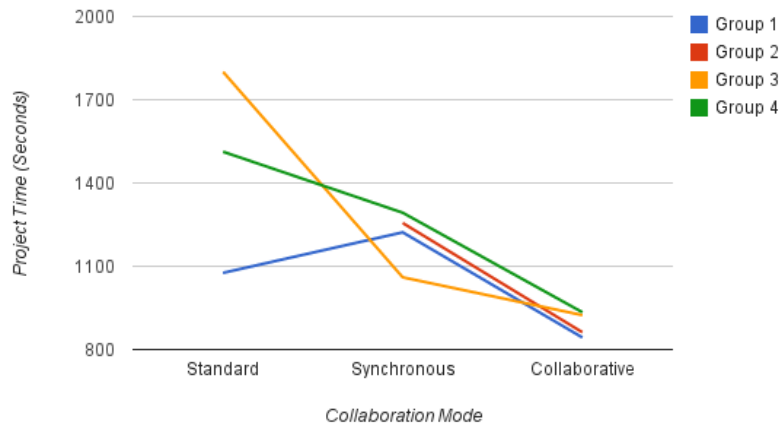


Figure 5.3: The total project times for Workflow A by group

Each group also experienced a learning curve, both for using Firestorm and for completing the project. This learning curve strongly influenced the completion times. Figure 5.4 shows the same data as Figure 5.3, but the columns have been rearranged to match the order of the testing sequence.

5.2.2 Qualitative Results

The qualitative results were obtained during tests on a collaborative workflow and from post-study discussions with the participants. This discussion begins with the key observations from the collaborative tests. The first important result from these tests deals with how people

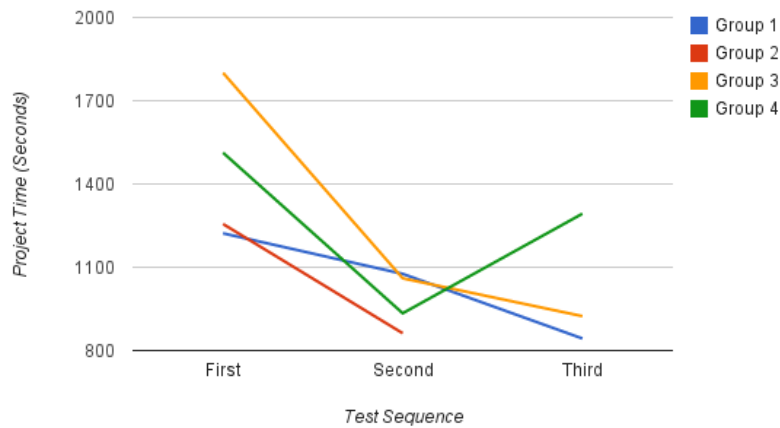


Figure 5.4: Project times for Workflow A by test order

chose to divide work at the workflow level, as well as at the document level. The process of dividing work always occurred in two phases: an initial phase, with a follow up adaptation phase. At the workflow level, the initial phase occurred whenever new documents became available. At the document level, the initial phase occurred when the document was blank. During this phase, the group decided on a general strategy. At the workflow level, this meant assigning people to work on specific activities. At the document level, this meant assigning pieces of the document (such as a paragraph of a text document, or a column of data in a spreadsheet). The team worked as assigned until an event triggered adaptation. Usually, the adaptation phase began because someone finished their portion; however, it also occurred when one task or portion appeared to be falling behind.

An example of this can be clearly seen from the collaborative workflow of Group 2. Figure 5.5 shows the progression from the initial phase to the adaptation phase. Once the first text document was completed, this team spent about one minute deciding how to divide the tasks. They chose the configuration shown in Figure 5.5a. They remained this way for about two minutes. Then, because the spreadsheet was quickly becoming a bottleneck, the triangle user chose to go to the spreadsheet, as shown in Figure 5.5b.

This behavior was also observed in the collaborative workflow of Group 1. The last text document and spreadsheet in the workflow became available at about the same time. The group decided to divide the work as shown in Figure 5.6a. They worked this way for a little over three

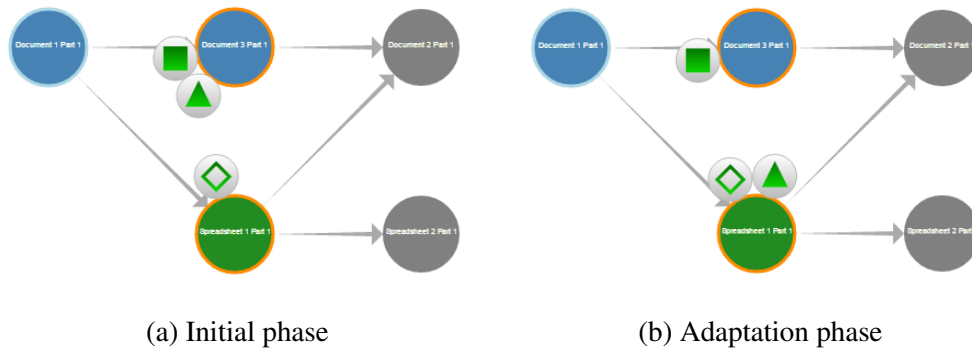


Figure 5.5: An example of the initial and adaptation phases of division of labor

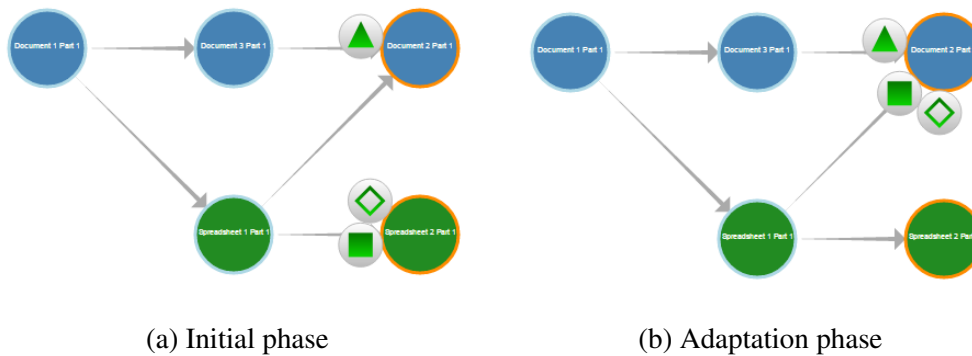


Figure 5.6: A second example of the initial and adaptation phases

minutes. Then, the square and diamond users finished the spreadsheet, and joined the triangle user to work on the document, as shown in Figure 5.6b.

Another important result was observed in Workflow A and in Workflow B. It dealt with dividing work based on the skills of the participants. In Group 1, for example, the triangle user worked exclusively on text documents, because they felt more comfortable with typing text than entering data. This also occurred in Workflow B. In this case, only the diamond user had any experience with CAD modeling. The group decided that the most effective approach would be to allow the diamond user to work on the CAD part alone while the other three worked on the spreadsheet, as shown in Figure 5.7.

When the groups were asked about their experience, all groups responded that the collaborative workflow mode was the most enjoyable. This was mainly due to the fact that the other two modes often forced users to wait for their activity to become available, without being able to do

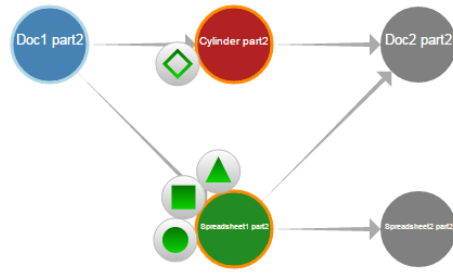


Figure 5.7: An example of skill-based division of labor

anything about it. A few people mentioned that they really liked being able to choose the work they felt comfortable doing. This sentiment was very strongly expressed by the group that executed Workflow B.

One interesting result was expressed by one of the volunteers, who suggested that the collaborative workflow was in some ways more demanding than the others. It required the user to continuously absorb and act upon the awareness information provided by others. In some ways, this was more difficult than accepting assignments, without managing collaboration with others.

5.3 Discussion

The quantitative results are encouraging. In general, a group took 17% less time using the synchronous workflow than using the standard workflow. It is difficult to analyze the exact extent to which synchronous applications improved the project time for many reasons. The order in which tests were conducted had a major influence on the project times. All groups were able to improve the project time from the first to the second test. The group that improved the least did the synchronous workflow first, followed by the standard workflow.

This does not mean however, that the learning curve was more influential on project time than synchronous applications. Another powerful factor influencing project time was individual skill. Because users could not choose which role they fulfilled, they often found themselves doing a task that required skills they did not possess. It is possible that the individuals in Group 1 were assigned to activities they found difficult in the first test, and then to activities they were good at for the second.

Skill alignment partially explains the extreme improvement in Group 3 from the first to the second test. Two of the members of this group were particularly bad at entering spreadsheet data, and these two were each assigned to a spreadsheet. The spreadsheet tasks, which took about six minutes for the other users, took more than 10 minutes for these users. During the second test, the third member of this group (who was more capable of working on spreadsheets) happened to be assigned to both spreadsheets, saving the team quite a bit of time.

The project times suggest that synchronous workflows are faster than standard workflows. Unfortunately, there are relatively few data points and many confounding variables, both of which make it difficult to statistically prove that synchronous workflows are in fact faster. More tests would have been conducted, but time constraints made it impossible. However, it is easy to observe that if all participants were proficient at every task, and more participants were allowed to work on each task, synchronous workflows would be much faster than standard workflows.

The project times also suggest that collaborative workflow are the fastest kind of workflow. Every group, regardless of workflow order, achieved the best results using the collaborative workflow. In fact, the worst performance on the collaborative workflow was better than the best performance from any other kind of workflow. This performance benefit is due to two main causes: skill-based adaptation, and priority-based adaptation. In the collaborative workflows, group members were able to coordinate their activities so that their skills aligned with the tasks. Skill-based adaptation was specifically observed in two of the collaborative tests for Workflow A, and it is unlikely that it was not part of the decision-making process for the other groups. Priority-based adaptation allowed users to rearrange themselves throughout the workflow. The activities that became bottlenecks varied from group to group, but in each case, the group was able to reduce the effect of the bottleneck. Examples of this benefit were shown in Figures 5.5 and 5.6.

An even more intriguing (and unanticipated) result is that the standard deviation of the collaborative workflow times is extremely low. The fastest group only beat the slowest group by 72 seconds, or a little over a minute. By comparison, the group that had the fastest standard workflow time beat the group with the slowest time by more than 720 seconds, or 12 minutes. The standard deviation of the standard workflow times is equivalent to 25% of the average time. For collaborative workflows, on the other hand, the standard deviation of project times is equivalent to only 5% of the average project time. This results stands in spite of variables like test order

and group skill. Once again, this improvement can be attributed to skill-based and priority-based adaptation.

Two things need to be clarified here. First of all, the time improvements (in both average and standard deviation) of collaborative workflows are due to the ability that each team member had to choose where to work. That choice is far more complicated and meaningful because of the presence of synchronous applications. Any team member can choose to help someone else, and they can make that choice without a scheduled collaborative session. Continuous awareness enables people to respond quickly to the dynamics of the workflow. The awareness required for this kind of collaboration is easily provided in collocated situations. When users are not collocated, that awareness must be provided by the workflow system.

Second, a free-for-all assignment strategy is not feasible for large projects with large teams. The point of testing that strategy is to demonstrate the ideal benefits of an integrated system with continuous awareness. Practical projects may require some combination of assignments and freedom (enabled by awareness), but no such system has been developed or tested.

Returning to a discussion of the results, it is important to understand the response of the participants to various kinds of collaboration. As might have been predicted, people enjoyed having the freedom to work where they wanted, referring to the collaborative workflow. This allowed them to use the skills they felt comfortable with to benefit the group. As one participant noted, this does make the work environment somewhat more demanding for each participant. However, the project times speak for themselves. In spite of the additional demand, all groups performed better when asked to manage themselves.

An integrated system like Firestorm has the capacity to dramatically reduce project times, through the use of synchronous applications and continuous awareness. An integrated system can also minimize the variation of project times, making projects more predictable and increasing the productivity of a company. More statistically rigorous research needs to be done to understand and model these benefits, but these preliminary results justify an optimistic view towards future research.

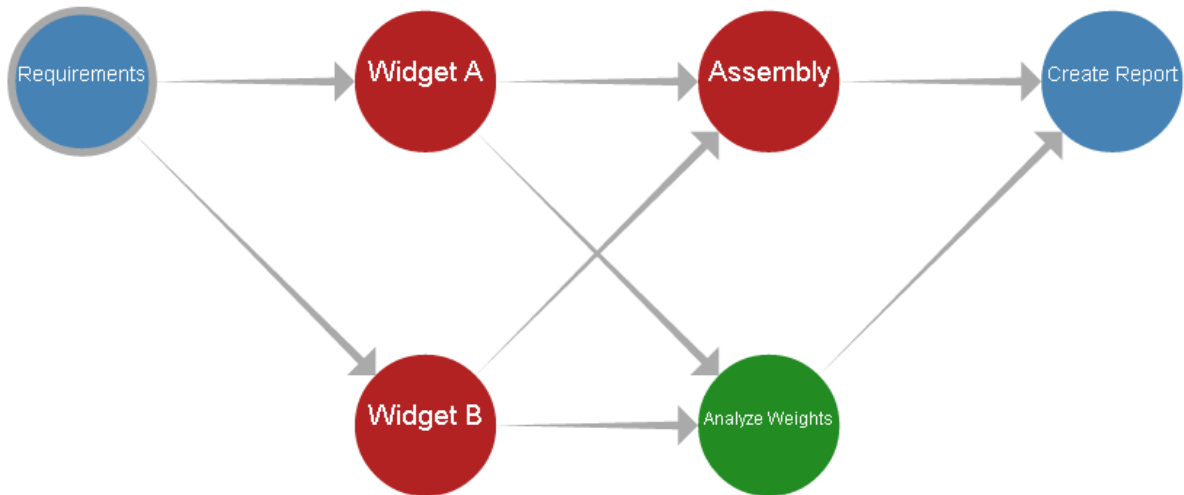


Figure 5.8: A simple engineering project

5.3.1 Shortcomings of the Study

The study reviewed here provides valuable results, and strongly encourages future studies in this area. However, the study also has several inherent shortcomings. First of all, activities did not truly depend on one another. This meant that users never had to refer to an activity's prerequisites in order to complete the task. Additionally, if a user were to open two documents simultaneously, Firestorm does not propagate awareness of that fact; users only appear to deal with a single document. Finally, the study did not involve any actual engineering tasks. This was mostly due to the shortage of volunteers.

A more meaningful study could involve a small project, driven by a simple scenario. It is important that this study involve as many varied tasks as possible, so that skill-based adaptation can be observed. It is also important that each activity truly requires the previous activity. To that end, the following scenario and workflow could be presented:

At a meeting, a client sketched a vague design for two parts which need to be assembled, and gave some requirements. These were recorded by hand. The purpose of the project is to take the sketch and requirements from a concept to a completed design. The workflow might look something like the one shown in Figure 5.8.

The first task, a document, would involve creating a formal requirements document based on the client's needs. The next two tasks would involve creating both parts as separate part files,

ensuring that they meet the requirements in the requirements document. Next, an assembly part could be created, defining constraints and relationships between the two parts. Simultaneously, a spreadsheet containing information about the parts could be created. Finally, a report, meant to be given to the client, could be generated.

A project of this nature more closely resembles an actual engineering project, and would likely force the users to refer to prerequisite documents. This study could be done in Firestorm, although Firestorm's awareness capacity might need to be updated to include awareness of multiple open documents per person.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

Based on the background research, as well as the testing that was performed, several conclusions can be drawn:

- Using synchronous applications can reduce project times. In the study conducted, using synchronous applications accounted for a 17% reduction in project time, even though some tasks were still completed by a single user.
- Using synchronous applications can also reduce the variation in project times. In the study conducted, variation was reduced from 25% of the project time to 8.5%.
- Allowing users to manage themselves, in addition to using synchronous applications, can reduce project times even further. In the study conducted, project times were reduced by a further 22%.
- Self management can also reduce the variation further. In the study conducted, variation was reduced to 5% of the project time.

In order to allow people to manage themselves at the project level, awareness must be present at the project level and the project must involve synchronous applications. Without awareness, self management would be utterly impossible because no one would have enough information to make meaningful decisions. Without synchronous applications, there are very few meaningful decisions to make beyond simple person/task assignments, which are easily automated. When both awareness and synchronous applications are present, however, people can make meaningful and helpful decisions for themselves.

Once people can manage themselves, the total time to complete a project can be reduced, beyond the time savings provided by synchronous applications alone. As bottlenecks arise during project execution, the team can easily adapt to relieve that bottleneck. Because of awareness,

everyone understands where help needs to be given. Because of synchronous applications, people can jump in and help when they need to. Self management helps reduce the effect of variables like individual skill or experience. It also reduces the effect of unforeseen delays. If any task begins to take longer than anticipated, either because of skill or some other issue, the team can adapt to relieve the bottleneck, which reduces the project completion time as well as the variation in project completion times.

It is obvious from the time-saving nature of synchronous applications that they should be integrated with PLM systems. However, the benefits of self management are powerful and should not be overlooked. PLM systems should seek to obtain, whenever possible, the benefits of self management. They can do this by providing awareness at the project level and by allowing people to work without a specific assignment. Allowing self management, even occasionally, will help engineering companies meet the demands they face.

6.2 Future Work

In the course of conducting this research, several areas for further study presented themselves. These subject include the following:

- The effect of the integrated system on project times
- The use of the integrated system in more complicated projects
- Handling synchronous and standard applications in the same workflow
- Modifying check-in/check-out for a commercial PLM system
- Leadership roles in the integrated system
- Assignments in the integrated system
- Scaling awareness
- Work Monitoring

6.2.1 The Effect of the Integrated System on Project Times

The most immediate work that could be based on this research is the study of project times in the integrated system. The results from Chapter 5 suggest that the integrated system reduces not only the total time, but also the variability of the time. If project times became more

predictable, an entire company's performance could be improved. Determining the exact effect of both synchronous applications and workflow awareness on project times would be extremely valuable.

6.2.2 The use of the integrated system in more complicated projects

In the case of more realistic projects, several new possibilities arise. For example, someone might want a requirements document and a CAD part open at the same time. Future research might deal with how to represent a person working in multiple documents. This situation arises naturally when a task truly depends on its prerequisites. A project like the one described in Section 5.3.1 might be well suited to this purpose. Additionally, further testing would provide additional data-points, allowing for a more rigorous statistical analysis.

6.2.3 Handling Synchronous and Standard Applications

It is unlikely that a workflow will ever require synchronous applications exclusively. While synchronous applications are very powerful, they are not always the best solution. For example, software development requires asynchronous work; otherwise, testing a single user's changes would be impossible. In addition, synchronous applications have not been developed for many, many engineering tasks. For the foreseeable future, workflows will use both standard and synchronous applications. Determining how to collaborate effectively using both standard and synchronous applications is essential.

6.2.4 Modifying Check-in/Check-out

Another necessary field of research deals with modifying the check-in/check-out system that all PLM systems have. This research would first require a far better understanding of that system and the other features related thereto. Once that understanding is achieved, methods for modifying that system to handle synchronous applications could be explored.

6.2.5 Leadership Roles in the Integrated System

This research considered a free-for-all approach to leadership at the workflow level. While this approach was simple to implement, it is unlikely to be helpful on actual projects. It would therefore be useful to study how leadership can be handled in this environment. This study could also include how to increase individual accountability in a totally multi-user environment. Ideally, this would be done without removing the benefits of workflow awareness. This work will be critical to the successful adaptation of the method.

6.2.6 Assignments in the Integrated System

This research also considered a free-for-all approach to assignments. Every member of the team was responsible for every assignment. This approach is also unlikely to scale to commercial projects. This work could possibly study a hybrid of assignments and self-direction. Again, the ideal situation would be to introduce the organization of assignments without destroying the benefits of workflow awareness. This work is also required before the method can be successfully adopted.

6.2.7 Scaling Awareness

It would also be beneficial to study how to handle awareness data on a very large project. There may be a point where the sheer quantity of awareness information becomes unhelpful and even detrimental. It would be very helpful to understand how to provide only the awareness a person needs, especially on a large project.

6.2.8 Work Monitoring

One of the downsides of allowing individuals to manage themselves is that not all people seek out work. Some people might use the lack of direct management to shirk responsibility, allowing others to complete the majority of the work. As the number of people working on a project grows, and as teams become more geographically dispersed, it becomes more and more

difficult to understand the contributions of any given individual. It might be possible for someone to appear busy, when in reality they are contributing very little to the overall project.

Therefore, it would be beneficial to research some kind of work monitoring method. Each client computer could somehow monitor (perhaps through key and mouse events) the activity of the user. Algorithms for detecting minimal or fraudulent activity data would need to be developed, the end goal being that a manager could be made aware of an individual who is not performing. In addition, the manager could also be made aware of those who are performing exceptionally well, and of the strengths and weaknesses of every individual.

REFERENCES

- [1] Red, E., Jensen, C., Ryskamp, J., and Mix, K., 2011. “NXConnect: Multi-User CAx on a Commercial Engineering Software Application.” In *PACE Faculty Forum*. 4, 12, 34
- [2] Stark, J., 2011. *Product Lifecycle Management*. Springer. 5
- [3] Batenburg, R., Helms, R. W., and Versendaal, J., 2005. “The maturity of Product Lifecycle Management in Dutch organizations A strategic alignment perspective.” *Institute of Information and Computing Sciences, Utrecht University*, pp. 1–14. 7, 8
- [4] Lee, S. G., Ma, Y. S., Thimm, G. L., and Verstraeten, J., 2008. “Product lifecycle management in aviation maintenance, repair and overhaul.” *Computers in Industry*, **59**, pp. 296–303. 8
- [5] Philpotts, M., 1996. “An introduction to the concepts, benefits and terminology of product data management.” *Industrial Management & Data Systems*, **96**, pp. 11–17. 8
- [6] Gascoigne, B., 1995. “PDM: the essential technology for concurrent engineering.” *World Class Design to Manufacture*, **2**, pp. 38–42. 8
- [7] Wang, Z., ter Hofstede, A., and Ouyang, C., 2014. “How to guarantee compliance between workflows and product lifecycles?.” *Information Systems*. 8
- [8] Hill, S., 2003. How To Be A Trendsetter: Dassault And IBM PLM Customers Swap Tales From The PLM Front. 8
- [9] Hollingsworth, D., and Hampshire, U., 1993. “Workflow management coalition the workflow reference model.” *Workflow Management Coalition*. 8
- [10] Lawrence, P., 1997. “Workflow handbook 1997.”. 8
- [11] Anand, M. K., Bowers, S., Altintas, I., and Ludwiger, B., 2010. “Approaches for exploring and querying scientific workflow provenance graphs.” In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 6378 LNCS, pp. 17–26. 9
- [12] Silva, C., Freire, J., and Callahan, S., 2007. “Provenance for visualizations: Reproducibility and beyond.” *Computing in Science & Engineering*. 9
- [13] Callahan, S., and Freire, J., 2006. “VisTrails: visualization meets data management.” In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 745–747. 9
- [14] Lousa, M., Sarmiento, a., and Machado, a., 2000. “Expectations towards the adoption of workflow systems: the results of a case study.” *Groupware, 2000. CRIWG*, pp. 36–41. 9

- [15] Siller, H. R., Estruch, A., Vila, C., Abellan, J. V., and Romero, F., 2008. “Modeling work-flow activities for collaborative process planning with product lifecycle management tools.” *Journal of Intelligent Manufacturing*, **19**(6), June, pp. 689–700. 9
- [16] Dourish, P., and Bellotti, V., 1992. “Awareness and Coordination in Shared Workspaces.” In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, no. November, pp. 107–114. 9, 10, 11
- [17] Greenberg, S., and Marwood, D., 1994. “Real time groupware as a distributed system.” In *Proceedings of the 1994 ACM conference on Computer supported cooperative work - CSCW '94*, ACM Press, pp. 207–217. 9, 11
- [18] Gutwin, C., and Greenberg, S., 1996. “Workspace awareness for groupware.” In *Conference companion on Human factors in computing systems common ground - CHI '96*, ACM Press, pp. 208–209. 9
- [19] Schmidt, K., 2011. *Cooperative Work and Coordinative Practices.*, Vol. 5 of *Computer Supported Cooperative Work* Springer London, London. 10
- [20] Greenberg, S., Gutwin, C., and Cockburn, A., 1996. “Awareness through fisheye views in relaxed-WYSIWIS groupware.” *Graphics interface*. 10
- [21] Ellis, C. A., Gibbs, S. J., and Rein, G., 1991. “Groupware: some issues and experiences.” *Communications of the ACM*, **34**(1), Jan., pp. 39–58. 11
- [22] Gutwin, C., and Greenberg, S., 2002. “A Descriptive Framework of Workspace Awareness for Real-Time Groupware.” *Computer Supported Cooperative Work (CSCW)*, **11**(3-4), Sept., pp. 411–446. 11
- [23] Dekeyser, S., and Watson, R., 2006. “Extending google docs to collaborate on research papers.” *Toowoomba, Queensland, AU: The University of Southern Queensland, Australia*. 12
- [24] Blau, I., and Caspi, A., 2009. “What type of collaboration helps? Psychological ownership, perceived learning and outcome quality of collaboration using Google Docs.” In *Proceedings of the Chais conference on instructional technologies research*. 12
- [25] Tan, X., and Kim, Y., 2011. “Cloud Computing for Education: A Case of Using Google Docs in MBA Group Projects.” *2011 International Conference on Business Computing and Global Informatization*, pp. 641–644. 12
- [26] Bidarra, R., van den Berg, E., and Bronsvort, W. F., 2001. “Web-based collaborative feature modeling.” In *Proceeding of the 6th ACM symposium on Solid modeling and applications*, pp. 319–320. 12
- [27] Mix, K., Jensen, C., and Ryskamp, J., 2010. “Automated design rationale capture within the CAx environment.” *Computer-Aided Design and Applications*. 12
- [28] Nysetvold, T., and Teng, C., 2013. “Collaboration tools for multi-user CAD.” *2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 1–5. 12

- [29] Hepworth, A. I., Tew, K., Nysetvold, T., Bennett, M., and Greg Jensen, C., 2014. “Automated Conflict Avoidance in Multi-user CAD.” *Computer-Aided Design and Applications*, **11**(2), Mar., pp. 141–152. 12
- [30] Hepworth, A., Tew, K., Trent, M., Ricks, D., Jensen, C. G., and Red, W. E., 2014. “Model Consistency and Conflict Resolution With Data Preservation in Multi-User Computer Aided Design.” *Journal of Computing and Information Science in Engineering*, **14**(2), Mar., p. 021008. 12
- [31] Weerakoon, P., Wu, J., Bright, T., Teng, C. C., Red, E., Jensen, G., and Merkley, K., 2013. “A networking architecture for a multi-user FEA pre-processor.” *Computer-Aided Design and Applications*, **10**(February 2015), pp. 527–540. 12
- [32] Briggs, J. C., 2014. “Developing an Architecture Framework for Cloud-Based, Multi-User, Finite Element Pre-Processing.” Master’s thesis, Brigham Young University. 12
- [33] Briggs, J. C., Hepworth, A. I., Stone, B. R., Coburn, J., Jensen, C. G., and Red, E., 2015. “Integrated, Synchronous Multi-User Design and Analysis.” *Journal of Computing and Information Science in Engineering*(c). 13
- [34] Red, E., Jensen, G., Weerakoon, P., French, D., Benzley, S., and Merkley, K., 2013. “Architectural Limitations in Multi-User Computer-Aided Engineering Applications.” *Computer and Information Science*, **6**(4), Aug., p. p1. 13
- [35] Ben-Shaul, I., and Kaiser, G., 1996. “Integrating groupware activities into workflow management systems.” *Proceedings of the Seventh Israeli Conference on Computer Systems and Software Engineering*, pp. 140–149. 13
- [36] Weber, M., Partsch, G., and Höck, S., 1997. “Integrating synchronous multimedia collaboration into workflow management.” *Proceedings of the Eighth International Workshop on*, pp. 281–290. 13
- [37] Li, F. L. F., Lin, Z. L. Z., Guo, Y. G. Y., and Li, J. L. J., 2001. “A workflow system for supporting group activities of an enterprise.” *Proceedings of the Sixth International Conference on Computer Supported Cooperative Work in Design (IEEE Cat. No.01EX472)*, pp. 440–443. 13
- [38] Rubart, J., Haake, J. M., Tietze, D. a., and Wang, W. “Component-Based Cooperative Hypermedia.” *Components*, pp. 73–82. 14
- [39] Wang, W., Finch, K., Rubart, J., and Haake, J. M., 2009. “a Cooperative Hypermedia Approach To Flexible Process Support for Managing Distributed Projects.” *International Journal of Cooperative Information Systems*, **18**, pp. 481–512. 14
- [40] Sun, D., Xiong, X., Ruan, L., Liu, Z., Zhao, J., and Wong, Y., 2004. “Workflow-driven collaborative session management in product lifecycle management via Internet.” In *2004 IEEE International Engineering Management Conference (IEEE Cat. No.04CH37574)*, Vol. 3, IEEE, pp. 1146–1150. 14, 26
- [41] Flatt, H. P., and Kennedy, K., 1989. Performance of parallel processors. 27

- [42] Xu, Z., and Hwang, K., 1996. “Modeling communication overhead: MPI and MPL performance on the IBM SP2.” *IEEE Parallel and Distributed Technology*, **4**(1), pp. 9–23. 27
- [43] Demeuleester, E., Herroelen, W., 1992. “Procedure for the Multiple Resource-Constraint Project Scheduling Problem.” *Management Science*, **38 SRC** -(12), pp. 1803–1818. 27
- [44] Kolisch, R., 1996. “Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation.” *European Journal of Operational Research*, **90**(2), pp. 320–333. 27
- [45] Merkle, D., Middendorf, M., and Schmeck, H., 2002. “Ant colony optimization for resource-constrained project scheduling.” *IEEE Transactions on Evolutionary Computation*, **6**(4), pp. 333–346. 27
- [46] Gustafson, J. L. “Reevaluating Amdahl’s Law.”. 28

APPENDIX A. THE FIRESTORM SERVER AND CLIENT

A.1 Complete Source Code

The complete source code is available through two sources. The author keeps a personal copy; for access, please send inquiries to j.lowellj@gmail.com. In addition, a backup is kept on a server managed by BYU's PACE lab. The complete path to the source code, including the server name, is `\\newgomer\nxconnect\Backups\JordanJohnsonFirestorm`.

A.2 Background

In order to test the method presented in Chapter 3, the Firestorm prototype was created. The prototype was created strictly for testing purposes. The following considerations drove the design of Firestorm:

- Firestorm needed to provide an integrated system, as described in Chapter 3.
- Firestorm's purpose was to be a testing platform. While it also fits the description of an integrated system, its primary purpose was to aid in test administration, data collection, and data analysis.
- Firestorm needed to be flexible. When Firestorm was created, the exact testing procedure had not been formulated. Firestorm was designed to allow for many kinds of tests.
- Firestorm needed a simple update system for bug fixes or feature changes. Since Firestorm required the use of many computers, it needed to be easy to propagate changes to all computers.
- Firestorm needed to be easy for anyone to use. This would allow non-technical people to participate in testing.
- Firestorm did not need to be very performance-friendly. Because tests would involve no more than 6 simultaneous users, performance was not a significant design consideration.

It became obvious that a web application would be an excellent way to handle these considerations. A web application allows the client to be browser-based, which simplifies installation and maintenance. Web applications also allow multiple users to access a service simultaneously, a core requirement of the integrated system. For these reasons, Firestorm was created as a web application.

In order to create Firestorm, a wide range of libraries and tools were used. The end functionality of Firestorm could feasibly be achieved using different tools and approaches. However, the tools chosen were familiar to the author and provided the necessary features.

All libraries, software, and services used were legally obtained free of charge.

A.3 Firestorm: a Django Web Application

Firestorm was developed using a free web framework called Django 1.7, the latest release at the time. Django is a Python-based web framework that simplifies the creation of web applications. It provides excellent object relational mapping (ORM) tools, along with easy-to-use routing methods. Django is also very well documented. For further information, visit www.djangoproject.com.

A.3.1 Environment

Because Django is based in Python, development was done using a virtual machine running Ubuntu 14.04, a Linux-based operating system. The virtual machine was managed by VMWare Player, a free machine emulator. The Python libraries were installed within a virtual environment created with `virtualenv`. More information and free downloads can be found at the following websites:

- VMWare - www.vmware.com
- Ubuntu - www.ubuntu.com
- Python - www.python.org
- Virtualenv - virtualenv.pypa.io/en/latest/

```

class User(models.Model):
    CODE_NAMES = (
        ('TRI', 'Triangle'),
        ('SQR', 'Square'),
        ('DIA', 'Diamond'),
        ('CRC', 'Circle'),
        ('RNG', 'Ring'),
        ('ADM', 'Admin')
    )
    name = models.CharField(max_length=3, choices=CODE_NAMES)
    nxcUsername = models.CharField(max_length=20)
    nxcPassword = models.CharField(max_length=20)
    admin = models.BooleanField(default=False)

```

Figure A.1: The User model for Firestorm

A.4 The Database and Server

Because of the close link between the data for Firestorm and the logic applied to that data, they will be discussed together in this section. Firestorm used PostgreSQL (www.postgresql.org) as a database system. However, all database interaction was done through Django's ORM features. Django uses a Python object called a *model* to represent a database object. For example, a user in Firestorm is represented by the Python class shown Figure A.1.

This class inherits Django's Model class. Each model class is represented by a table, and each instance of the class is represented by a row in that table. The fields in the table are represented with Django-specific types (such as `models.CharField` and `models.ForeignKey`). By defining an object this way, Django can create and manipulate a table of these objects.

A.4.1 Workflows

At a high level, the database represents test results. Firestorm represents two kinds of tests: creation tests and execution tests. Both kinds of tests involve a representation of a workflow and a list of events that modified that workflow. Workflows are represented by the Python class shown in Figure A.2.

Figure A.2 does not show any methods that exist on the Workflow class. Also, Django adds members to a class that reflect object relationships. In addition to the members shown in Figure A.2, Django also added the following members:

```

class Workflow(models.Model):
    STANDARD = 0
    SYNCHRONOUS = 1
    COLLABORATIVE = 2

    MODEL_TYPES = (
        (STANDARD, 'Standard'),
        (SYNCHRONOUS, 'Synchronous'),
        (COLLABORATIVE, 'Collaborative'),
    )

    name = models.CharField(max_length=255)
    model = models.IntegerField(choices=MODEL_TYPES)
    team = models.ForeignKey(Team, null=True)
    valid = models.BooleanField(default=False)
    referenced = models.BooleanField(default=False)

```

Figure A.2: The Workflow model for Firestorm

- `activity_set` - a collection of all activities in the workflow
- `assignment_set` - a collection of all assignments in the workflow
- `link_set` - a collection of all links in the workflow

These members are included because the Activity, Assignment, and Link classes (all of which inherit Django's Model class) have a foreign key that refers to the workflow to which they belong. A workflow, therefore, consists of users (a team), activities, assignments, and links. A diagram is shown in Figure A.3. Note that activities also have a position, which is used purely for drawing purposes. Figure A.4 shows a workflow with 5 links, 5 assignments, and 8 assignments.

A.4.2 Workflow Logic

The Workflow object provides logic for manipulating workflows. Assignments, activities, and links are added through methods defined by the Workflow object. These methods ensure that items are added correctly, and that the workflow ends up being valid.

Notice that mode of the workflow (indicated by the *model* variable) is a fundamental attribute of the workflow. The mode affects assignment logic, as shown in Figure A.5. In standard mode, only one person can be assigned to an activity. In synchronous mode, multiple people can

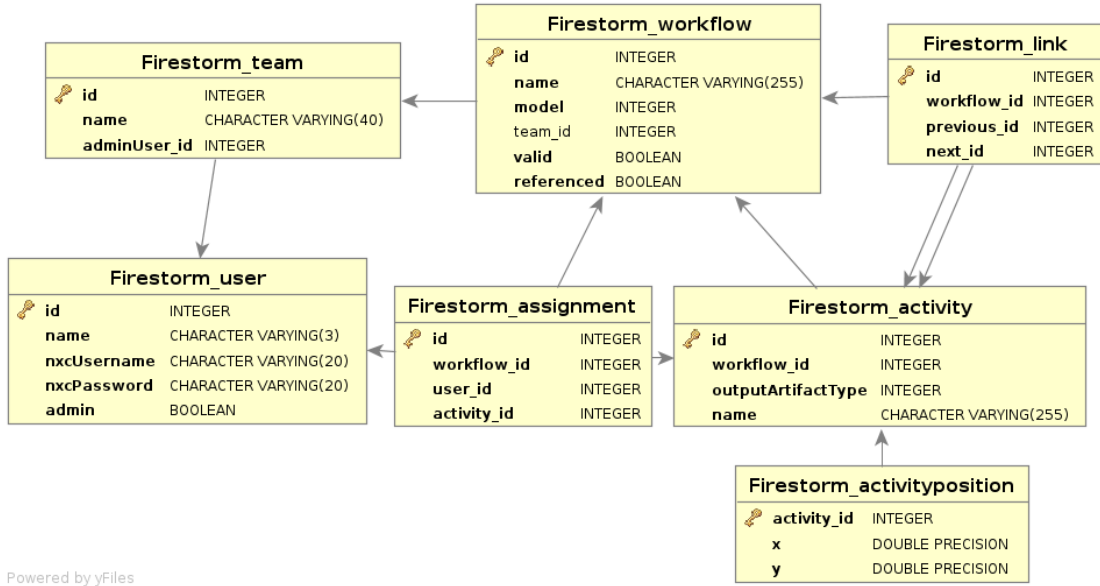


Figure A.3: A diagram of the object relationships defining a Workflow

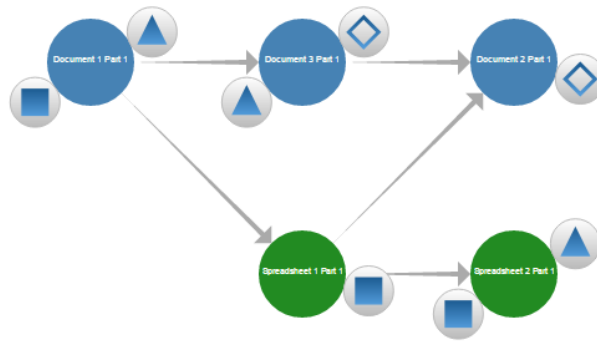


Figure A.4: An example workflow

be assigned to an activity. There are no assignments in collaborative mode. In no case can a user be assigned to an activity more than once.

Activities can be added at any time. Firestorm defines an activity as a task with one associated document, called an artifact. The artifact can only be a Google Doc, a Google Sheet, or an NXConnect part.

Links can be added between two existing activities. However, when a link is added, the Workflow enforces several rules. First of all, there can only be one link between any two activities, and only in one direction. Also, a link cannot form a cycle in the workflow. Finally, a check for

```

def addAssignment(self, user, activity):
    # no assignments can be made for collaborative mode
    if self.model == Workflow.COLLABORATIVE:
        return

    # check for activity existence in this workflow
    try:
        activity = self.activity_set.get(pk=activity.pk)
    except Activity.DoesNotExist:
        return

    # person assigned only once to an activity
    try:
        self.assignment_set.filter(user=user).get(
            ↪ activity=activity)
        return
    except Assignment.DoesNotExist:
        pass

    # standard mode – one person per activity
    if self.model == Workflow.STANDARD:
        if self.assignment_set.filter(activity=activity).
            ↪ count() >= 1:
            return

    assignment = Assignment()
    assignment.workflow = self
    assignment.user = user
    assignment.activity = activity
    assignment.save()

    self.valid = False
    self.save()

    return assignment

```

Figure A.5: The Workflow addAssignment method

redundant links is made (if there is a link from *a* to *b* and from *b* to *c*, there shouldn't be a link from *a* to *c*).

A workflow must also reference a team, where a team is a collection of users. In Firestorm, there are 5 possible users (Triangle, Square, Diamond, Circle, Ring, and Admin). These users are organized into 3 teams, as outlined below. These teams (and the users on those teams) were guaranteed to exist in the database using the Django fixture feature.

- Team 1 - Triangle, Square, Diamond
- Team 2 - Triangle, Square, Diamond, Circle
- Team 3 - Triangle, Square, Diamond, Circle, Ring

Finally, a workflow must be *validated*, by calling the `validate()` method on the workflow. This method traverses the structure of the workflow and ensures that there is only one project defined by the nodes and links. This means that there cannot be multiple unrelated sets of activities and links; all activities must be linked into a single tree.

The rules for link creation and workflow validation are meant to keep the workflows relatively simple. They also form a logical foundation that the execution tests depend on to function properly.

A.4.3 Workflow Creation Tests

A `WorkflowCreationTest` object represents the process that created a workflow in Firestorm. A test is composed of a workflow, a tester (a first and last name - who 'took the test'), comments, and a set of `WorkflowCreationEvent` objects.

When the test is created, it creates a corresponding workflow. Then, `WorkflowCreationEvent` objects are applied. Each event object represents an action that affects the workflow. They each have a type, enumerated below:

- `CREATION_BEGUN` - The creation test is begun
- `CREATION_ENDED` - The creation test is over
- `ACTIVITY_ADDED_DOC` - The user added a Document activity
- `ACTIVITY_ADDED_SPD` - The user added a Spreadsheet activity

- `ACTIVITY_ADDED_CAD` - The user added a CAD activity
- `ACTIVITY_NAME_SET` - The user changed the name of an activity
- `ACTIVITY_REMOVED` - The user removed an activity
- `ACTIVITY_MOVED` - The user updated the viewing position of the activity
- `LINK_ADDED` - The user added a link between two activities
- `LINK_REMOVED` - The user removed a link
- `TEAM_SELECTED` - The user selected a team
- `ASSIGNMENT_ADDED` - The user added an assignment to an activity
- `ASSIGNMENT_REMOVED` - The user removed an assignment
- `WORKFLOW_CHECKED` - The user tried to validate the workflow
- `WORKFLOW_SUBMITTED` - The user submitted the workflow (can only be done after successful validation)
- `WORKFLOW_CANCELED` - The user canceled the workflow

Each event also contains specific information about that event, using up to 4 optional fields. As long as the test has not received the `CREATION_ENDED` event, it can receive and apply additional events. Applying a given set of creation events to an empty workflow is a determinant operation. The method by which these events are generated and handled will be discussed later.

A.4.4 Workflow Instances

Once a workflow has been validated, it can then be *instantiated*. The idea behind instantiation is to use a workflow as a template. When a workflow is instantiated, a `WorkflowInstance` object is created. For each activity in the workflow, an `ActivityInstance` object is created, along with a corresponding artifact. Artifact creation will be discussed shortly. A `LinkInstance` object is also created for each link. Assignment instances are not necessary, because `ActivityInstance` objects store a reference to the `Activity` object they were instantiated from.

A `WorkflowInstance` object is just like its parent workflow, except that it is capable of changing its state. Each `ActivityInstance` object creates an artifact. In the case of Google Doc and Google Sheet activities, this is a fairly straightforward process, but requires a Google service


```

@staticmethod
def authenticate():
    SERVICE_ACCOUNT_EMAIL = [some service email account name]
    SERVICE_ACCOUNT_PKCS12_FILE_PATH = [a private key file
        ↪ path]
    f = file(SERVICE_ACCOUNT_PKCS12_FILE_PATH, 'rb')
    key = f.read()
    f.close()
    credentials = SignedJwtAssertionCredentials(
        ↪ SERVICE_ACCOUNT_EMAIL, key,
            scope='https://www.googleapis.com/auth/drive')
    http = httplib2.Http()
    try:
        http = credentials.authorize(http)
        return build('drive', 'v2', http=http)
    except Exception as e:
        print e
        return None

```

Figure A.6: Creating and authenticated Drive Service object

account¹. The WorkflowInstance object authenticates a Google Drive service object, and then passes that object to each WorkflowInstance so that they can use it to create either a Doc or a Sheet, depending on their artifact type. This process is shown in Figures A.6 and fig:createGoogleDoc.

In this way, the database can create and provide access to Google Docs and Sheets. The database does not directly deal with NXConnect parts, although the file name for each part is generated at this time.

A.4.5 Workflow Execution Tests

A WorkflowExecutionTest object represents the changing state of a WorkflowInstance object as users interact with it. It is composed of a WorkflowInstance, a collection of WorkflowExecutionUserEvent objects, and a collection of WorkflowExecutionModel event objects. User events represent actions that any user can take in Firestorm. These are described below:

- USER_CLICKED - A user clicked on an activity node
- USER_DBL_CLICKED - A user double clicked on an activity node

¹Google service accounts are accounts meant to be manipulated by an application. These accounts can send emails and create unlimited drive documents (up to 250,000 a day), including Docs and Sheets. To create a project, and from there a corresponding service account, visit console.developers.google.com

```

def instantiate(self, workflowInstance, service):
    serviceGood = service is not None
    path = 'INVALID'
    fileId = 'INVALID'
    if self.outputArtifactType == ArtifactType.DOCUMENT and
        ↪ serviceGood:
        driveFile = self.makeGoogleDriveFile(
            service,
            workflowInstance,
            'application/vnd.google-apps.document'
        )
        path = driveFile['alternateLink']
        fileId = driveFile['id']

    elif self.outputArtifactType == ArtifactType.SPREADSHEET
        ↪ and serviceGood:
        driveFile = self.makeGoogleDriveFile(
            service,
            workflowInstance,
            'application/vnd.google-apps.spreadsheet'
        )
        path = driveFile['alternateLink']
        fileId = driveFile['id']

    else:
        # These will be created by a second request
        path = self.buildName(workflowInstance) + '.prt'
        fileId = self.name

    artifact = Artifact()
    artifact.artifactType = self.outputArtifactType
    artifact.workflowInstance = workflowInstance
    artifact.fileURL = path
    artifact.fileId = fileId
    artifact.save()

    instance = ActivityInstance()
    instance.workflowInstance = workflowInstance
    instance.artifact = artifact
    instance.activity = self

    instance.save()

    return instance

```

Figure A.7: The Activity::instantiate method

- `USER_MARKED_COMPLETE` - A user attempted to mark an activity as complete
- `USER_MARKED_INCOMPLETE` - A user attempted to mark an activity as incomplete
- `USER_ENTERED` - A user entered the test (by navigating to the test page)
- `USER_EXITED` - A user exited the test (by navigating away from the test page)
- `USER_KILLED` - A user (an administrator) stopped the test
- `USER_CHAT` - A user sent a chat message to the group

User events are accompanied by any relevant information (which user, which activity, etc.). When the `WorkflowExecutionTest` object receives a new user event, it analyzes the previous user and model events. These events essentially store the state of the `WorkflowInstance` object. Using the state information, the test can determine what, if any, model events to generate as a response to the user event. Model events are only generated in response to user events. They include the following kinds of events:

- `VISITED` - A user has visited an activity
- `OPENED` - A user opened the document associated with an activity
- `COMPLETED` - An activity was successfully marked complete
- `INCOMPLETED` - An activity was successfully marked incomplete
- `FINISHED_ALL` - Everyone who needed to mark an activity complete did so
- `KILLED` - The test has been successfully canceled

If an event changes the status of an activity, that status can be changed to one of the following values:

- `NEW` - No one has opened the document yet
- `IN_PROGRESS` - At least one person has opened the document
- `COMPLETE` - Everyone who needed to mark this activity complete did so

Like `WorkflowCreationEvent` objects, `WorkflowExecutionModelEvents` contain many optional fields that are used to fully describe the event. Depending on the type of event, various fields are expected to contain values. The `WorkflowExecutionModel` event is defined by the Python class shown in Figure A.8:

```

class WorkflowExecutionModelEvent(models.Model):
    VISITED = 1
    OPENED = 2
    COMPLETED = 3
    INCOMPLETED = 4
    FINISHED_ALL = 5
    KILLED = 6
    ACTIONS = (
        (VISITED, 'visited'),
        (OPENED, 'opened'),
        (COMPLETED, 'completed'),
        (INCOMPLETED, 'incompleted'),
        (FINISHED_ALL, 'finished'),
        (KILLED, 'killed'),
    )

    NEW = 1
    IN_PROGRESS = 2
    COMPLETE = 3
    STATUS = (
        (NEW, 'new'),
        (IN_PROGRESS, 'in_progress'),
        (COMPLETE, 'complete')
    )

    workflowExecutionTest = models.ForeignKey(
        ⇨ WorkflowExecutionTest)
    userEvent = models.ForeignKey(WorkflowExecutionUserEvent)
    user = models.ForeignKey(User, related_name='
        ⇨ spawning_user')
    activity = models.ForeignKey(ActivityInstance, null=True)
    online = models.NullBooleanField()
    targetUser = models.ForeignKey(User, null=True,
        ⇨ related_name='targeted_user')
    action = models.IntegerField(choices=ACTIONS, null=True)
    accessible = models.NullBooleanField()
    status = models.IntegerField(choices=STATUS, null=True)

```

Figure A.8: The WorkflowExecutionModelEvent class

```

def isAccessible(self, user=None):
    if user is None:
        q = Q(targetUser__isnull=True)
    else:
        q = Q(targetUser=user) | Q(
            ↪ targetUser__isnull=True)
    q = q & Q(activity=self) & Q(accessible__isnull=
        ↪ False)
    selfEventList = WorkflowExecutionModelEvent.
        ↪ objects.filter(q)
    selfEvent = selfEventList.last()
    if selfEvent is None:
        return False

    return selfEvent.accessible

```

Figure A.9: The ActivityInstance::isAccessible class

When a user event reaches the server, the server determines which (if any) model events to generate. This logic is shown in Section A.4.6; all methods shown belong to the WorkflowExecutionUserEvent object. All user events are handled by the respondToEvent() method, which forwards the events on to an appropriate handler.

The workflow state can be determined by querying the list of events. For example, in order to determine whether a given user has access to a given activity, the method shown in Figure A.9 is used. It gets a list of all WorkflowExecutionModelEvent objects that apply to the given user for this activity, and finds the last one where the accessible field is not null, returning that value.

All other queries about the state of the model are handled in a similar fashion: get the list of events that apply to a particular activity or user, and determine the state of the model from that list.

A.4.6 Model Event Handling

```

@staticmethod
def respondToEvent(userEvent):
    online = userEvent.user.isOnline(userEvent.
        ↪ workflowExecutionTest)
    # if the user is not online, they can't interact with the
        ↪ model
    if userEvent.event == WorkflowExecutionUserEvent.
        ↪ USER_ENTERED:

```

```

        if not online:
            WorkflowExecutionUserEvent.respondToEnter
                ↪ (userEvent)
        return

# admins can do whatever they want when offline
# everyone else doesn't even get to keep this event
if not (online or userEvent.user.admin):
    userEvent.delete()
    return

if userEvent.event == WorkflowExecutionUserEvent.
    ↪ USER_KILLED:
    WorkflowExecutionUserEvent.respondToKilled(
        ↪ userEvent)

if userEvent.event == WorkflowExecutionUserEvent.
    ↪ USER_EXITED:
    WorkflowExecutionUserEvent.respondToExit(
        ↪ userEvent)

if userEvent.event == WorkflowExecutionUserEvent.
    ↪ USER_CLICKED:
    WorkflowExecutionUserEvent.respondToClick(
        ↪ userEvent)

if userEvent.event == WorkflowExecutionUserEvent.
    ↪ USER_DBL_CLICKED:
    WorkflowExecutionUserEvent.respondToDbClick(
        ↪ userEvent)

if userEvent.event == WorkflowExecutionUserEvent.
    ↪ USER_MARKED_COMPLETE:
    WorkflowExecutionUserEvent.respondToMarked(
        ↪ userEvent)

if userEvent.event == WorkflowExecutionUserEvent.
    ↪ USER_MARKED_INCOMPLETE:
    WorkflowExecutionUserEvent.respondToUnmarked(
        ↪ userEvent)

# chat messages don't affect the model

@staticmethod
def respondToClick(userEvent):
    """ The user has clicked on an activity """
    """ Generates one event for everyone that the user
        ↪ visited a node """

```

```

modelEvent = WorkflowExecutionModelEvent()
modelEvent.apply(userEvent)
# modelEvent is null
# targetUser is null
modelEvent.action = WorkflowExecutionModelEvent.VISITED
# accessible is null
# status is null

modelEvent.save()

@staticmethod
def respondToDbClick(userEvent):
    """The user is attempting to open an artifact"""
    # check if accesible
    if not userEvent.activity.isAccessible(userEvent.user):
        return

    # check permissions
    if userEvent.workflowExecutionTest.workflowInstance.
        ↪ workflow.model != Workflow.COLLABORATIVE:
        if not userEvent.activity.isAssigned(userEvent.
            ↪ user):
            return

    # check to see if we need to change the status
    curStatus = userEvent.activity.getStatus()
    if curStatus == WorkflowExecutionModelEvent.NEW:
        curStatus = WorkflowExecutionModelEvent.
            ↪ IN_PROGRESS

    modelEvent = WorkflowExecutionModelEvent()
    modelEvent.apply(userEvent)
    modelEvent.action = WorkflowExecutionModelEvent.OPENED
    modelEvent.status = curStatus

    modelEvent.save()

@staticmethod
def respondToMarked(userEvent):
    """The user has marked the activity as complete"""
    # check if accesible
    if not userEvent.activity.isAccessible(userEvent.user):
        return

    # check permissions
    if userEvent.workflowExecutionTest.workflowInstance.
        ↪ workflow.model != Workflow.COLLABORATIVE:

```

```

        if not userEvent.activity.isAssigned(userEvent.
            ↪ user):
            return

# check that it is in progress
if userEvent.activity.getStatus() !=
    ↪ WorkflowExecutionModelEvent.IN_PROGRESS:
    return

# check if it's already been marked complete by this user
if userEvent.activity.hasUserMarkedComplete(userEvent.
    ↪ user.id):
    return

modelEvent = WorkflowExecutionModelEvent()
modelEvent.apply(userEvent)
modelEvent.targetUser = userEvent.user
modelEvent.action = WorkflowExecutionModelEvent.COMPLETED
modelEvent.save()

# check if this is the last needed complete event
if userEvent.activity.haveAllAssignedMarkedComplete():
    # change the status
    modelEvent = WorkflowExecutionModelEvent()
    modelEvent.apply(userEvent)
    modelEvent.status = WorkflowExecutionModelEvent.
        ↪ COMPLETE
    modelEvent.save()

# update accessibility
userEvent.workflowExecutionTest.workflowInstance.
    ↪ updateActivityAccess(userEvent)

# Check if everything is complete
if userEvent.workflowExecutionTest.
    ↪ workflowInstance.isFinished():
    modelEvent = WorkflowExecutionModelEvent
        ↪ ()
    modelEvent.apply(userEvent)
    modelEvent.activity = None
    modelEvent.action =
        ↪ WorkflowExecutionModelEvent.
        ↪ FINISHED_ALL
    modelEvent.save()

```

```

@staticmethod
def respondToUnmarked(userEvent):

```



```

""" The user has marked the activity as incomplete """
# check if accesible
if not userEvent.activity.isAccessible(userEvent.user):
    return

# check permissions
if userEvent.workflowExecutionTest.workflowInstance.
    ↪ workflow.model != Workflow.COLLABORATIVE:
    if not userEvent.activity.isAssigned(userEvent.
        ↪ user):
        return

# check if it's already completed
if userEvent.activity.getStatus() ==
    ↪ WorkflowExecutionModelEvent.COMPLETE:
    return

# make sure this user has marked this activity as
    ↪ complete in the first place
if not userEvent.activity.hasUserMarkedComplete(userEvent
    ↪ .user.id):
    return

# check if this user has already marked incomplete
if userEvent.activity.hasUserMarkedIncomplete(userEvent.
    ↪ user.id):
    return

modelEvent = WorkflowExecutionModelEvent()
modelEvent.apply(userEvent)
modelEvent.targetUser = userEvent.user
modelEvent.action = WorkflowExecutionModelEvent.
    ↪ INCOMPLETED
modelEvent.save()

```

@staticmethod

```

def respondToEnter(userEvent):
    modelEvent = WorkflowExecutionModelEvent()
    modelEvent.apply(userEvent)
    modelEvent.online = True
    modelEvent.save()

```

@staticmethod

```

def respondToExit(userEvent):
    modelEvent = WorkflowExecutionModelEvent()
    modelEvent.apply(userEvent)
    modelEvent.online = False
    modelEvent.save()

```

```
@staticmethod
def respondToKilled(userEvent):
    modelEvent = WorkflowExecutionModelEvent()
    modelEvent.apply(userEvent)
    modelEvent.action = WorkflowExecutionModelEvent.KILLED
    modelEvent.save()
```

A.5 The Firestorm Interface

The Firestorm interface is a series of web pages provided by the Firestorm server. These web pages fall into several categories, as follows:

- A home page, providing navigation to the other pages
- Creation pages, related to creation tests
- Execution pages, related to execution tests
- Project pages, describing the various tasks to be completed during testing
- Training pages, providing information to new users on how to use Firestorm
- Review pages, related to reviewing completed tests

Most of the data on these pages is static, and can easily be comprehended by viewing it. A discussion of the important dynamic data follows.

A.5.1 The Execution Test Interface

By far the most complicated and important interface in Firestorm is the execution test interface. This interface shows a live view of a given workflow. It automatically updates based on the actions of all users involved with the test. This is accomplished, at a high level, in the following steps:

1. Visit the execution test page (a GET request). This page is dynamically generated. At this phase, the server creates a page for a given execution test and user. This page makes automatic requests, and the URLs for those requests are set when the page is rendered by Django.

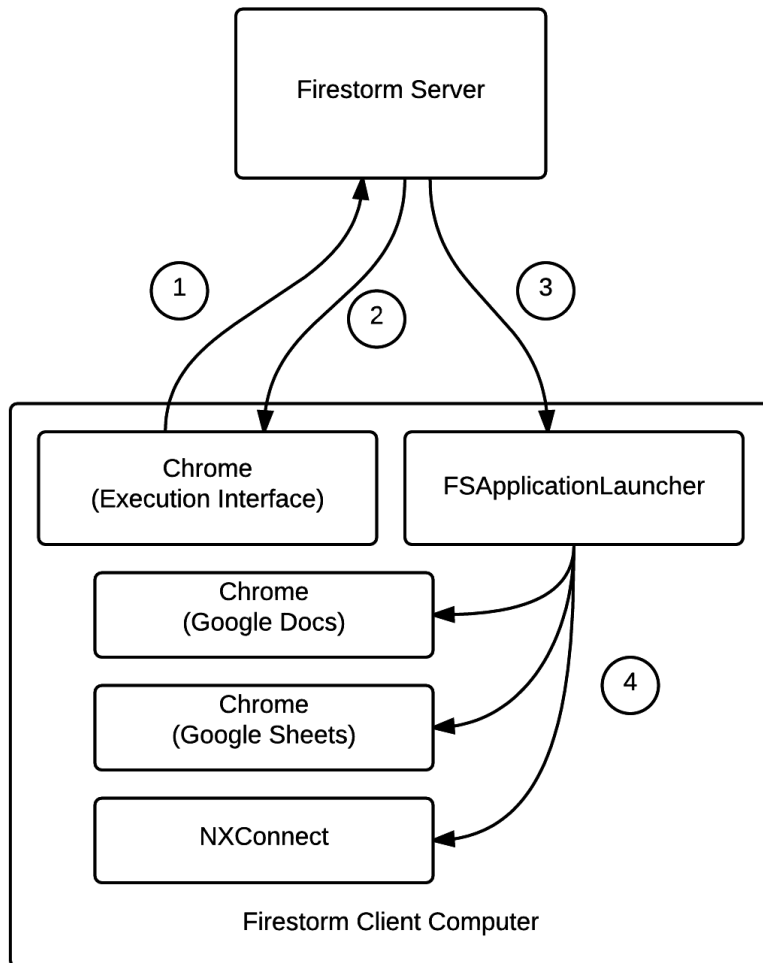
2. Acquire the workflow model. This happens when the execution page is loaded. The page (using JavaScript/ajax calls) requests a JSON version of a specific workflow. As a response to this request, the database returns a JSON object representing the workflow, which is parsed into a JavaScript object. Every client receives a clean version of the workflow initially.
3. Begin the event loop. This happens after the workflow is loaded and parsed. At a set interval, the following steps occur:
 - The page requests all events that have occurred since the last received event. This includes user events and model events.
 - In response to this request, the server provides a JSON version of all the events the page needs. These events are parsed into JavaScript objects.
 - Each event is applied to the workflow. The client-side version of the workflow has a changeable state, so that queries about the state do not require a search through the database.
 - The last received event is updated.

This model behaves such that a user's own changes are not applied immediately on the client. Rather, changes are sent (via POST request) to the server immediately for verification. They are not applied until they are received from the server.

A.5.2 The FSApplication Launcher

An important component of the execution interface is the FSApplicationLauncher. One of the more convenient features of Firestorm is the ability to open a document in a specific application, simply by double clicking on an activity. JavaScript does not provide a way to launch external applications. In order to provide this functionality, each client computer runs an instance of the FSApplicationLauncher. This enables a document to be opened as shown in Figure A.10.

This same forwarding mechanism also allows Firestorm to instantiate CAD parts. When a workflow is instantiated, the server creates the WorkflowInstance object and the related documents. Once this is complete, the server forwards a list of parts that need to be created to the FSApplicationLauncher, which passes that list to NXConnect. NXConnect creates the parts and closes immediately.



1. The Firestorm server receives and verifies an open document request
2. The server records the event, which is soon retrieved by the client
3. The server forwards the request to the FSApplicationLauncher
4. The FSApplicationLauncher opens the documents in the correct application

Figure A.10: Opening a Firestore document

The final purpose of the FSApplicationLauncher was to make sure clients were still connected. At a very slow interval, each Firestorm interface would send a “heart beat” message to the server, which was forwarded to the FSApplicationLauncher. If a client failed to send the message for a long time, the server would send a message to the server, taking the client user offline.

In order for this system to work, the Firestorm server had to be able to connect to each client’s FSApplicationLauncher. Therefore, within Firestorm, a specific port became the FSApplicationLauncher port. The Firestorm server assumes that an FSApplicationLauncher is listening on the client computer at that specific port.

Additionally, because the FSApplicationLauncher occasionally makes POST requests, it must be able to find the Firestorm server. In order to facilitate this, the Firestorm server (the machine itself) was given a static IP address, and the network administrators were able to add a DNS entry, so that any computer on the network could get to the Firestorm server using the URL `firestorm.com`. A special URL was set up on the Firestorm server that allowed the FSApplicationLauncher to acquire a CSRF token, which Django requires for PUSH requests.

A.5.3 Workflow Visualization

The visualization of the workflow (including the locations of the user icons) uses the JavaScript library D3 (`d3.js`). This library makes it very easy to bind visual elements (SVG elements, in this case) to data. Each activity is bound to an SVG circle element. Each link is bound to a custom SVG path element, defining the link shape. Each user is bound to an svg-defined user icon. D3 makes it very easy to update the location of the visual element based on the data bound to that element.

Another important feature of Firestorm is that the user icons travel from activity to activity, which happens whenever a user clicks on a new activity. One of D3’s features is that it can create a particle simulation; usually, this is used to lay out very complicated graphs. D3 also allows for an easy implementation of collision algorithms. In Firestorm, the user icons are attracted to a given activity node. However, they also collide with activities and with each other. In this way, user icons always end up where they are supposed to be, and there is some awareness of where user is going.

A.5.4 Possible Improvements

There are several possible improvements to this system. The first is that it is not truly possible to track when a user closes a document, or even whether that user already has the document already open. This means it is entirely possible for users to open the same document twice. It also means that statistics about how long a particular document is open cannot be captured. The difficulty in tracking this arose with Google Chrome. When a Chrome process launches, it searches for another Chrome process. If it finds one, it somehow sends that process a message to open a new tab, and then promptly dies. The master Chrome process then spawns a new Chrome process that manages the new tab. The author was unable to find a reliable method for tracking these processes.

In addition, it seems there is a lot of code duplication between Python and JavaScript. It would be beneficial for development purposes to reduce this duplication.

Finally, the JavaScript library `prototype.js` was used to create more class-like objects. In the end, however, this library did not offer any more functionality than regular function-objects. In addition, the library conflicted with `jQuery`, which was very inconvenient. Further use of this library is discouraged.