Simulation as a Decision-Support Tool in Construction Project Management

by

Muhtasim Fuad Rafid

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Simulation is a powerful tool, critical to effective construction project management. They can provide decision support for different aspects of a construction project, but they have not been adopted widely in the construction industry as they are difficult to use.

This thesis has developed Simphony-as-a-Service, an advanced simulation tool that bridges the gap between simulation software and real-world construction practices. The key contributions of this work are the following: Re-architecting Simphony, a powerful multi-functional legacy software, into a cloud-based software as a service on the micro-service architecture for easy adoption; Grounding the tool in a project-modeling language aligned with industry standards; Providing a browser-accessible user interface, tailored to the needs of different stakeholders, i.e., project owners, project managers, and on-site personnel; Incorporating critical-path computation and comparisons to assist in identifying critical tasks and evaluating alternative schedules; Providing APIs for real-time project progress tracking, enhancing monitoring capabilities, and enabling model re-simulation from any point in time of a project for exploring what-if scenarios and facilitating informed decision-making.

Preface

All work presented in this thesis was done by Muhtasim Fuad Rafid under the supervision of Dr. Eleni Stroulia.

To my lovely wife Sarah

For her unwavering love and support

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Eleni Stroulia, for her continuous guidance and support throughout my graduate studies. She has always been kind to me and given me the best advice in all circumstances.

I would also like to thank Stephen Hague for assisting me with certain technical parts of this research and Samuel Iwuchukwu for helping me with the migration process.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the realm of construction management, the coordination of tasks like planning, scheduling, and budgeting involves a diverse team of engineers, architects, project managers, construction workers, and support staff [1]. A good schedule is critical for managing complex construction projects, to ensure that they deliver the desired products, on time, within budget, and using the planned resources. The construction schedule, i.e., a description of the project activities, their dependencies, the resources they require, and the time they may take, is indispensable for effective project management, from the inception of the project to its completion and delivery.

Project Management Software (PMS) tools provide techniques, such as Work Breakdown Structure, Gantt Charts, Critical Path Method, Simulation, and Resource Levelling, to assist Project Managers with scheduling and planning. A study by Galloway et al. surveyed construction personnel from the US and found out that PMS was mainly used for planning and scheduling before and during construction, subcontractor coordination, and tracking changes [2]. Microsoft Project [3] and Primavera P6 [4] are two of the most popular project-management software tools [2]. Microsoft Project, currently in its 16th version, supports project scheduling, report creation, what-if analysis, and interactive tasks. Primavera P6 provides similar features, including project timelines, risk management, information dashboard, work breakdown structure, and critical path management. Despite their rich functionalities, these tools are expensive and designed for very large-scale projects which makes their user interfaces fairly unwieldy. Even more importantly, they suffer

from two key shortcomings: they do not easily support learning from past project experiences and they restrict progress tracking to a single individual, responsible for being aware of all the details of all ongoing activities and providing the relevant information through the tool's standard user interface.

Simulation can be a powerful tool that can mitigate the first shortcoming. Several simulation tools have been proposed for the construction domain, like CYCLONE [5], STROBOSCOPE [6], and Simphony [7]. By using discrete event simulation to simulate the various project activities, these tools can incorporate the manager's experience in developing the construction schedule. For example, they can incorporate different statistical distributions in estimating the activity progress or the occurrence of events that can obstruct progress altogether. Therefore, they can be more effective when construction projects are complex, and environmental factors, like weather, risk, and accidents, should be taken into account. Despite this important potential advantage, a study by Abourizk et al. [8] found that simulation tools are not popular in the construction industry.

The second issue identified above becomes evident as the complexity and size of the project crew increase, when quite often the projects cannot follow the original schedule and adjustments are required. In these cases, the persons in the best position to accurately reflect progress and delays are the activity crew leads: they are much closer to the activity and the issues that are causing the delays than the project manager. It is essential then to provide a simple, preferably mobile, tool so that more accurate information can be collected in a timely manner. The availability of accurate and timely information on how the actual progress diverges from the schedule is key for the project manager's decision-making process, who has many options to choose from. For example, they may be able to hire more people, schedule more shifts for the available crew, and pay them overtime, or even do nothing and pay the penalty associated with the schedule overrun. The feasibility, cost, and time demands of these alternative scenarios can again be estimated using simulation with more precision, allowing better decision support.

To bridge the gap between real-world construction practices and simulation software, we have developed a new, web-accessible version of the SIMPHONY [7]

simulation tool, Simphony-as-a-Service ($Simphony_{aaS}$), which advances the state-of-the-art in the following key dimensions.

- We have re-architected the original the latest version of the Simphony [9] software into a micro-service architecture-based software as a service. This new architecture enables the easy adoption of the tools through the web and also the development of a suite of user-interface views tailored to the needs of different roles (i.e., project manager, crew lead, project owner, ...) and contexts (i.e., office, construction site etc). A key addition to the original SIMPHONY software is a simple mobile interface to enable easy on-site data entry by crew leads responsible for an activity.

- This new version is grounded in a project-modeling language, aligned with the W3C standard Digital Construction Ontology (DiCO) [10], which enables interoperable construction data representation.

- We have created a REST API that allows real-time project progress updates to be persisted in our system to monitor a construction project regularly. This enables re-simulation of the project schedule from any point in time during construction by automatically updating the model based on the real progress data, making decision choices easier for project managers.

- We have developed a semantic repository, based on the DiCon, that enables the cross-referencing of the baseline simulation with re-simulations for the same project so that project owners and managers can explore multiple what-if scenarios and compare their corresponding crews, critical paths, and time-lines

Figure 1.1 shows from a high level how $Simphony_{aaS}$ incorporates project managers and crew leads into the simulation scenario to make simulation more accessible. The crew leads provide real-time data to our system, our system updates the model automatically using this data, and project managers can run simulations at any time of the project to see future predictions and make decisions. We can see

that no human intervention is needed for automated model updates, which effectively makes the simulation model more re-useable throughout the project.

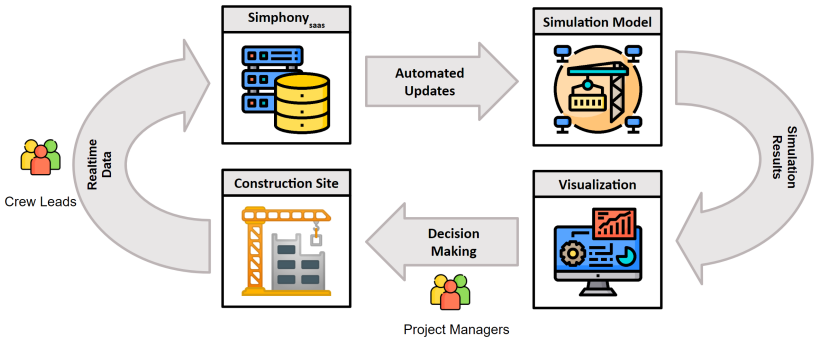


Figure 1.1: Incorporating users from different roles of construction in *Simphony*$_{aaS}$

The rest of this thesis is organized as follows. Chapter 2 describes the current literature on simulation construction. Chapter 3 discusses the simulation specification model and its alignment with DiCo. Chapter 4 describes the *Simphony*$_{aaS}$ architecture and an overview of the software system. Chapter 5 provides a walkthrough of the system through an illustrative example. In Chapter 6, we show the performance evaluation results. We discuss future work and conclude the thesis in section 7.

Chapter 2

Literature Review

2.1 Simulation in Construction - An Overview

The field of construction simulation encompasses a diverse range of techniques and methodologies aimed at modeling and simulating various aspects of construction projects. From scheduling and resource allocation to logistics and risk management, simulation provides a versatile framework for evaluating different scenarios and assessing their potential impact on project outcomes. Discrete event simulation, in particular, has been widely researched as a way to simulate the construction process. Research in construction-specific discrete-event simulation started with the introduction of CYCLONE in 1977 by Haplin [5], marking the inception of modern construction simulation languages. CYCLONE models, shown by a typical tunneling process in Figure 2.1, typically encompass elements for representing work tasks, their logical relationships, and resources required by the work task. The CYCLONE simulation language is simple to understand, but modeling large processes through the language is tedious as the modeling elements describe the simulation scenario at a low level. Besides, there is no way to distinguish between different types of the same resources and give different behavior to the same type of resources. One has to create completely separate resources for this scenario which would make elements in the model redundant. CYCLONE was further improved to MicroCYCLONE [11] which packaged CYCLONE into a microcomputer-based program for ease of use.

Stroboscope, introduced in 1994 by Martinez and Ioannou, is a general-purpose

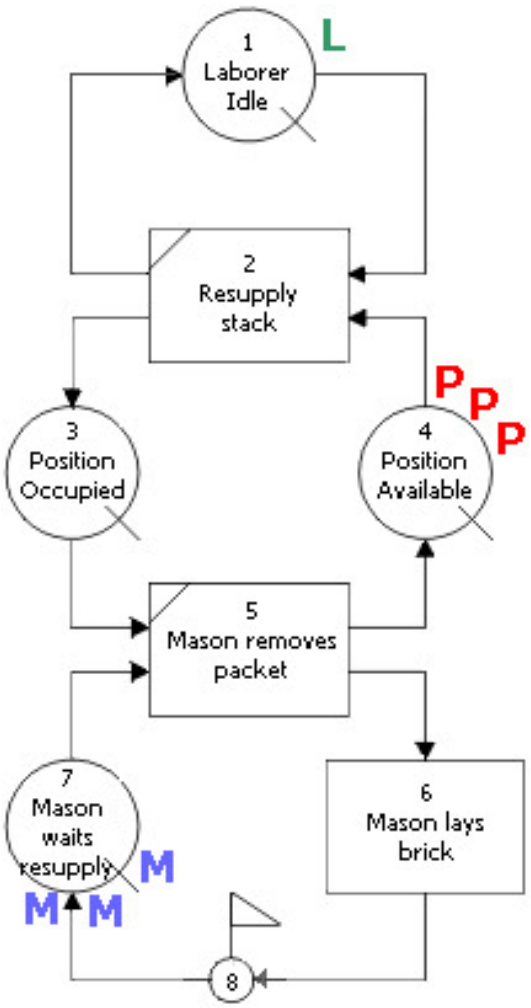


Figure 2.1: A Simple Masonry Supply System in the Cyclone model [12].

simulation programming language for the construction domain [6]. Driven by object-oriented programming, the language allowed the user to define activities, resources, and resource flow to create a simulation model. Later on, using this Stroboscope simulation engine, a special-purpose simulation tool called EarthMover [13] was created which had specialized elements for earth-moving simulation like excavation equipment, the excavation area, the loading capability of the equipment, and the queued-up load. It has a graphical user interface to define these parameters and outputs the results in an Excel sheet. Martinez also proposed another general-purpose simulation system called EZSTROBE [14] which is based on Activity Cycle Diagrams and uses the Three-phase Activity Scanning Algorithm. It uses 7 different elements to define the activity cycle as an input and outputs a report

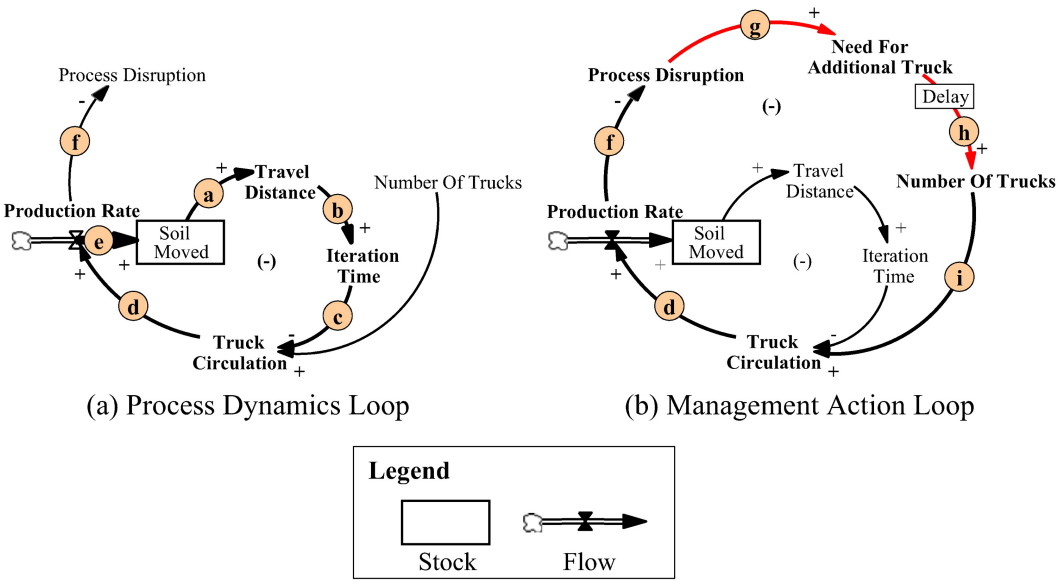
with the simulated timings and data on the activities and queues of the model.

In 1999, Simphony [7] was introduced as a discrete event, special-purpose simulation system for construction. It presented a modular and hierarchical modeling language to better represent larger and more complex projects. A graphical user interface or script-based interface both were available as means of model creation. Further development of Simphony and its current status is discussed in Section 2 of this chapter.

Modern research on construction simulation focuses on providing more meaningful insight for the construction industry. Lu and Olofsson [15] introduced a framework designed to facilitate the construction of simulation models, leveraging the enhanced functionalities offered by Building Information Modeling (BIM) systems. The system automatically reads data from BIM to create simulation models, which can then be run using the Simio [16] discrete event simulation engine.

Another study by Peña-Mora et al. [17] introduces a simulation model that integrates both the strategic and operational factors into account, optimizing both the process and management decision-making. It essentially creates a feedback loop within the simulation and updates construction operations based on management action within the simulation. The author created an earth-moving scenario and created a program that simulates the management decisions which is fed to the operational simulation when more or less trucks were needed. The study showed that management input changes the output of the simulation over time, depicting a more real-life scenario. However, the feedback loop exists only in the simulation, and management updates are provided through a program. So, there is no way to provide real data for the simulation. Figure 2.2 shows how this feedback loop works.

Feng et al. [18] in 2018 introduced a machine learning-based simulation approach. An artificial neural network was trained with the results produced from a discrete event simulation model. The author used this approach on a real project to demonstrate that it outperformed discrete event simulation in terms of prediction efficiency. However, it requires the ANN to be re-trained for each new model.



(a) Process Dynamics Loop (b) Management Action Loop

Figure 2.2: The simulation feedback loop of the system introduced by Peña-Mora et al. [17].

2.2 Simphony in Construction Simulation

Simphony [7], introduced in 1999, is a Microsoft Windows-based system that focuses on providing a standard, consistent, and intelligent environment for both the development as well as the utilization of construction special purpose simulation tools. Further development on Simphony was done to make it a more well-refined software package [19] with the capacity to craft specialized tools, known as templates, which can adapt to behaviors across various modeling languages, including integrated calendars for organizing tasks, discrete event simulation, and continuous simulation and introduces a variety of templates available in the public domain for both general and specialized modeling purposes such as tunneling, fabrication, earthmoving, and PERT. It is a system based on .NET Framework and provides 3 components, (1) Simphony Core Services - a discrete event simulation engine (2) Simphony Modeling Services - a collection of components that can be used to build graphical modeling elements for special and general-purpose simulation. (3) Simphony Modeling Environment, a graphical user interface that allows users to create simulation models using the graphical modeling elements. The platform provides robust support for seamlessly combining discrete event and continuous

simulation techniques. It offers the flexibility of incorporating calendars to synchronize simulation time with real-world dates, allowing for precise modeling of work periods and schedules. Additionally, the system offers versatile connectivity options, enabling seamless interaction with diverse data storage applications such as XML documents or databases. Moreover, it facilitates effortless integration into larger distributed simulation systems, including those compliant with the HLA standard, ensuring compatibility and interoperability within complex simulation environments. Overall, it was turned into a development kit for creating a special or general-purpose simulation system. Based on the 3 components of Simphony, a more construction-focused simulation methodology was proposed by Labban and Abourizk in 2021 [9]. It introduced construction terminologies like resources, crews, tasks, and products into the simulation language, which closely represents the construction process. It also introduced the concept of simulation based on real progress data, where simulation could be run from any point in time within the project based on real progress. However, updating the simulation model and collection of real data requires manual human intervention, limiting its capability.

2.3 Hindrance to Industry Adoption

Overall, construction simulation has evolved over the last 47 years to solve many different needs of the construction industry. However, most of it has still not been adopted by the construction industry [8]. A study by Lucko et al. [20] summarized the reason for this as - (1) The required extra time, cost, and effort to learn a new simulation language, model validation, and verification [21], (2) inefficient utilization of time for both the domain expert and the simulation analyst during the modeling procedure, (3) the advanced skills necessary for proficiently applying simulation modeling techniques [22]. In another study Son and Wysk state, “Unless the time-consuming phase of learning and using a simulation language is reduced, the advantages of simulation cannot be fully exploited” [23]. To solve these issues, Abourizk is collaborating with universities, industry, and government to effectively

integrate simulation into construction management functions in Alberta, and has found success [8]. The next section discusses how this success was achieved and what are the limitations that still need to be resolved. We focused our research on evolving the current Simphony simulation system into a modern software solution more suited for construction project management.

2.4 Dynamic, Data-Driven Simulation In Construction using Simphony - Achievements and Drawbacks

The new Simphony simulation system based on the proposal by Labban and Abourizk. [9] has been given the name 'Simphony Dynamic'. It has solved some of the major hindrances to the adoption of simulation in construction by introducing: (1) a modeling language that is aligned with the construction project management process, (2) an easy-to-use interface that enables model editing using drag-and-drop components and spreadsheet views, (3) a collection of visualizations to provide construction project management decision support. These made the system closer to adoption in the construction industry. A special version of Simphony Dynamic was created for PCL Construction, one of the largest construction companies in Canada, which added further features like (1) support for integrating historical task progress data into the simulation, (2) the addition of a baseline schedule that the simulation will adhere to and (3) introduced the concept of man hours to enable fractional engagement of resources to tasks.

However, we have identified multiple drawbacks that need to be solved for it to become a system more suited for the construction domain. Creating and updating a large simulation model using the graphical interface is tedious and lengthy due to the drag-and-drop procedure of building a task network, which can be solved by automating the model creation using data from the planning, design, or operational phase of construction. Multiple simulations of the same project from a different point in time require manual model parameter updates by the user. This can be solved by creating a method to automatically update the model parameters based on

the selected point in time. There is also no support for tracking real-time progress and re-simulation based on real-time progress automatically.

The software was built using .Net Framework 4.6 [24], which has entered maintenance mode and is now a legacy software framework, that needs to be migrated to the latest version, .NET 8.0 [25], to ensure future stability and maintenance.

The model and results are persisted using Microsoft Access DB in local files, so interoperability and data sharing is a challenge. There is no central data repository, so comparison and tracking of multiple simulation runs are difficult. Introducing a better database technology better suited for the simulation model and migrating the current AccessDB database will fix this issue.

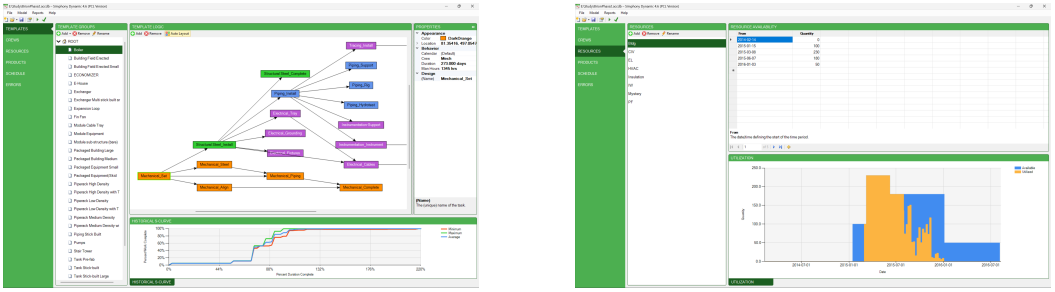


Figure 2.3: Graphical User interface of the Simphony Dynamic simulation software

In the rest of the thesis, we will discuss our efforts to solve these issues, how our system performs after we solve them, and put forward our contributions.

Chapter 3

Construction-Project Modeling

In this chapter, we discuss the representation of construction projects in *Simphony Dynamic* . In $Simphony_{aaS}$, we use the same model to describe construction projects.

3.1 The *Simphony Dynamic* Project-Specification Model

The original *Simphony Dynamic* project-specification model, diagrammatically shown in Figure 3.1 is based on well-understood, broadly used construction terminology for ease of use, and includes five main elements: *Tasks*, *Resources*, *Crews*, *W*orkflows, and *Products*.

Tasks are specific, well-defined units of work in the construction process. A construction project can be defined as a collection of tasks, connected with each other through dependencies. In order for the project to be completed, all its tasks must be completed in some order that respects the task predecessor-successor dependencies. The project *critical path* is one among these possible task orderings, which has the minimum possible length (in time).
Our model supports 3 types of dependencies between tasks:

1. Finish to Start - The predecessor ends before the successor can begin.
2. Start to Start - The predecessor begins before the successor can begin.
3. Finish to Finish - The predecessor ends before the successor can end.

The simulation also allows these dependencies to be partial i.e. only a fixed percentage of one task might need to be completed to start another task.

All *Task*s in Simphony have three important properties - *duration*, *manHours*, and *HistoricalProgress*. The *duration* property is the estimated duration needed to complete a task. The *manHours* property is the estimated man-hours needed to complete a task. The task's *HistoricalProgress* contains data on how this task progressed over time in previous projects, that can be used to create an S-curve to be provided as input to the simulation engine. If there is no such data from previous projects, the *HistoricalProgress* can be kept empty.

Resources are the manpower, and in some cases the equipment, required for the construction project. Usually, it is estimated when a particular resource will be needed and for how long at the beginning of a project, and the contract lengths of the resources are based on those estimations. For our simulation to understand these parameters, every *Resource* has an important property, a list of objects called *ResourceAvailability*. This object has two attributes - *availableFrom* and *quantity*. The *quantity* attribute indicates how many of a type of resource is available, and the *availableFrom* attribute indicates from when that *quantity* of resources can start working.

Crews are collections of different types of resources, required for a particular type of task. Each project task can only be accomplished if its corresponding crew is available. Each crew can perform multiple *Task*s of the same type, as long as they are not performed concurrently. For multiple concurrent same-type *Task*s an equal number of crews must be hired. Each *Crew* element has a list of objects called *CrewResources*. Each *CrewResources* object has two attributes - *quantity* and *resourceName*, to indicate which resource is attached to the crew and in what quantity.

In effect, the concept of "crews" encapsulates a set of individuals with a variety of skills and the equipment necessary for a specific task. Since projects typically have multiple repetitions of the same task, the task crew simplifies the need to assign

resources to each of these instances: the same crew can accomplish all repetitions, as long as they do not overlap in time. Realistically, the concept of crews represents a popular practice of maintaining, as much as possible, the same groups of people working together so that the project can benefit from their experience as a group, instead of having individuals getting accustomed to working in different groups all the time.

Workflows are collections of interconnected *Task*s that, when accomplished, produce a product, essentially achieving a(n intermediate) goal of the construction project. *Workflow*s, in effect, encapsulate stereotypical task collections that can be found in multiple projects. *Workflow*s in the Simphony system are reusable and can be saved to be used in multiple products of the same project. Thus, a construction project can be defined simply as a collection of interconnected *Task*s or as a collection of *Workflow*s and *Task*s. The *Workflow*s in a *Simphony Dynamic* project can be modified to adapt to the needs of the product.

Products are the outcomes of *Workflow*s. A product can be a physical product or a construction goal. For example, when building a multi-level building, there can be multiple *Workflow*s for multiple floors. It is possible to assign a *Workflow* to a product and customize that *Workflow* as suited for that particular product. This is done to remove the redundancy of creating the same or similar *Workflow* across multiple products of the same type.

When a *Workflow* is attached to a *Product*, a copy of all *Task*s and their connections are created and attached to the product. These new *Task*s are called *ProductTasks*. This enables dependency between *Task*s across products, where one *ProductTasks* might depend on a *ProductTasks* of another product. The properties of these elements are the same as *Task* with one additional property called *Schedule*, describing the estimated or planned start and end dates of the *ProductTasks*.

In $Simphony_{aaS}$, the project-simulation specification is represented as a graph in GraphDB and transmitted as a JavaScript Object Notation (JSON) [1] for commu-

[1] A sample Project-Specification Model JSON is available at: https://shorturl.at/fBCFX

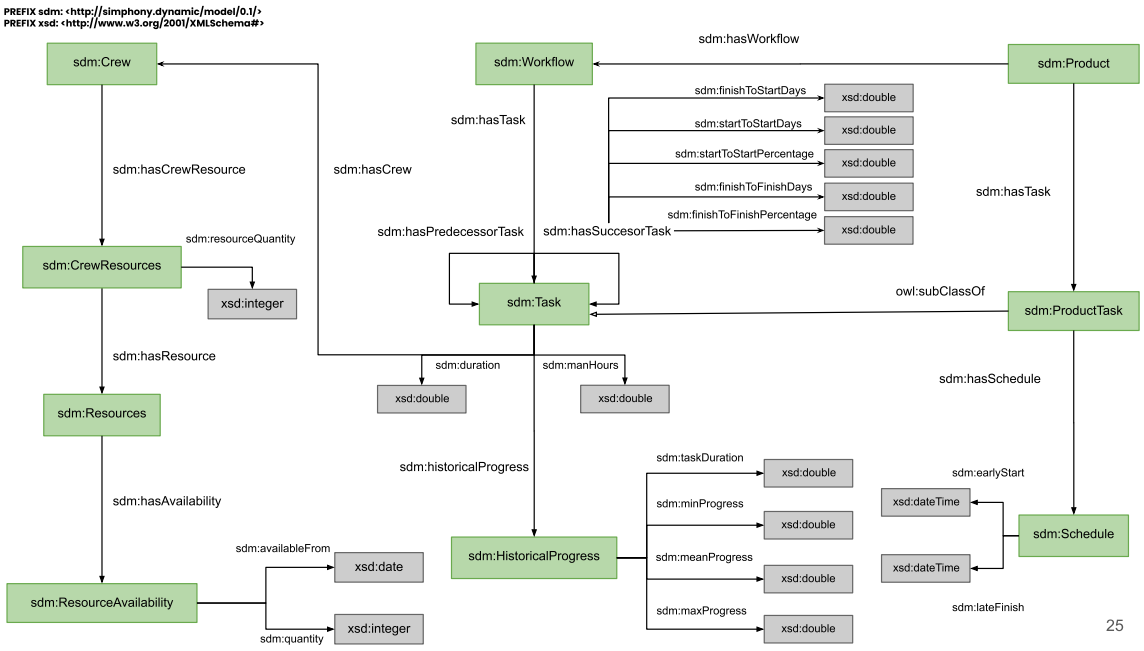


Figure 3.1: A diagram of the *Simphony Dynamic* project-specification model

nication over the network.

3.2 The Simulation Results

After the user sends a simulation request to $Simphony_{aaS}$, the simulation is run and the system returns a JSON [2] with all the necessary information for decision support. The results contain the calculated critical path, the progress of each task over time, the resource utilization over time, the crew utilization information (waiting times and file length), the progress of the tasks over time, and the estimated start and end date of each task. The overall structure of the simulation results is shown in Figure 3.2. We will discuss the results in detail in this section.

The Simulated Timings are the date and time when the tasks in the project were started and finished by the simulation. These results are directly attributed to the related *Task* of the project. Two data properties, called *simulatedStart* and *simulatedFinish* are linked to the task at the end of the simulation to represent the simulated start and end times.

[2] A sample simulation result JSON is available at: https://shorturl.at/rMTW8

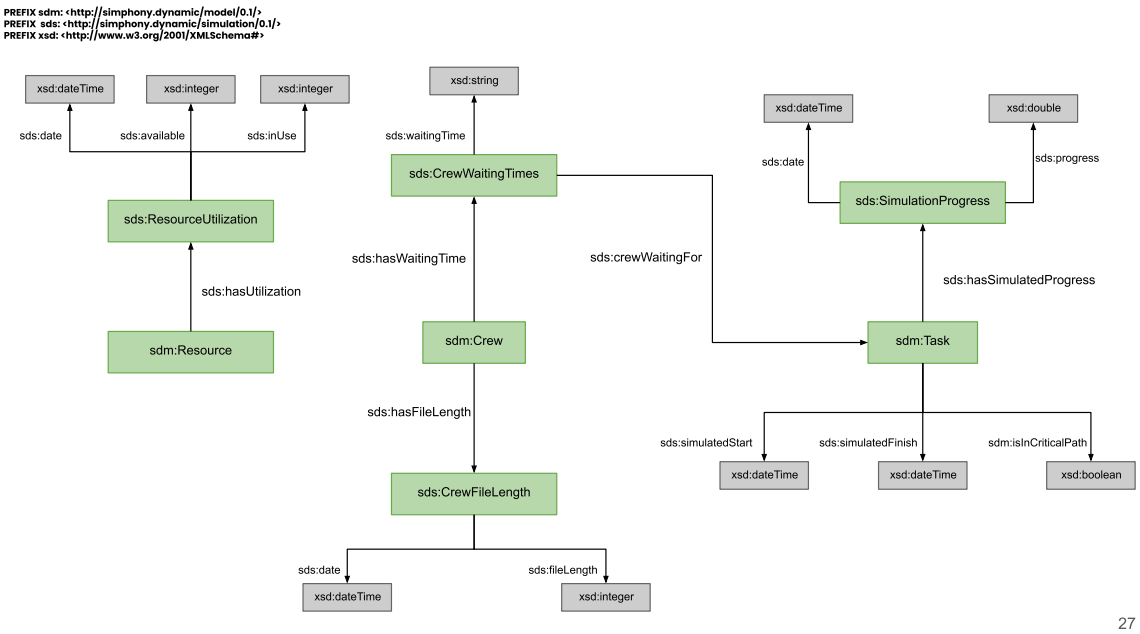


Figure 3.2: A diagram of the *Simphony Dynamic* simulation result model

The Critical Path Status of a task is represented by a boolean value attributed to the *Task*, called *isInCriticalPath*. If this value is true, the task is on the critical path.

The Task Progress Over Time is represented as a collection of objects called *SimulationProgress* linked to the relevant *Task*. Each object has 2 values: *date*, indicating on which date the progress was collected by the simulation, and *progress* , the percentage of work done on that date.

The Resource Utilization is represented by an object called *ResourceUtilization*, which has three attributes: *date*, *available* and *inUse*. The *date* attribute indicates on which date the simulation collected this data, *available* is the number of resources that were available on that day to the simulation and *inUse* indicates how many of the available resources were being utilized. A list of *ResourceUtilization* is attached a *Resource* to indicate which resource these results belong to.

The Crew Waiting Times show if crews were waiting for a task's dependencies to finish and were not being utilized. A *CrewWaitingTime* object represents this result and has only one attribute: *waitingTime*, which indicates the time span the

crew was waiting. The *CrewWaitingTime* is linked to a *Crew* and a *Task*, indicating which crew was waiting for which task.

The Crew File Lengths contains the number of crews that were waiting for *Task*s and were idle at any point in time of the project. It is represented using the *Crew-FileLength* object. The object has 2 properties: *date*, pointing to the date the crew was waiting, and *fileLength*, showing how many crews were waiting on that date. Each instance of a *CrewFileLength* object is linked to a *Crew*. One *Crew* object can be linked to multiple *CrewFileLength* objects.

3.3 Enabling Model Re-simulation and Reusability

As we have discussed in the literature review, Labban and Abourizk [9] proposed a system to incorporate automatic updates to the simulation model in Simphony. The updates were automatic in the sense that the productivity of each task is updated using Bayesian Inference and real data, but a person still has to input the real data. So to update the model, manual human interference is still required.

We wanted to achieve the same effect but without any human interference. For that, we created a REST API in the Repository microservice which would receive real progress data from the construction site and a mobile application through which crew leads would report the daily tasks progress. This enabled us to use the progress data straight from the database without manual intervention. Each real progress entry is saved as a *Real Progress* object, with 2 properties: *date*, indicating which date the progress entry was made and *progress*, the percentage of work done on that day. Each object is linked to a task in the project-specification model. Figure 3.3 shows *Real Progress* is linked to *Task* and its properties.

Whenever a user requests a simulation to our system from any point in time, we query the database to find out what was the progress of each task at that point. We then adjust the man hours and duration of the task based on that progress i.e. reduce them as required. For example, if 20% of a task was completed at the time of simulation, we changed the duration of the task to 80% of the original duration and 80% of the original man hours in the model in memory, without changing the

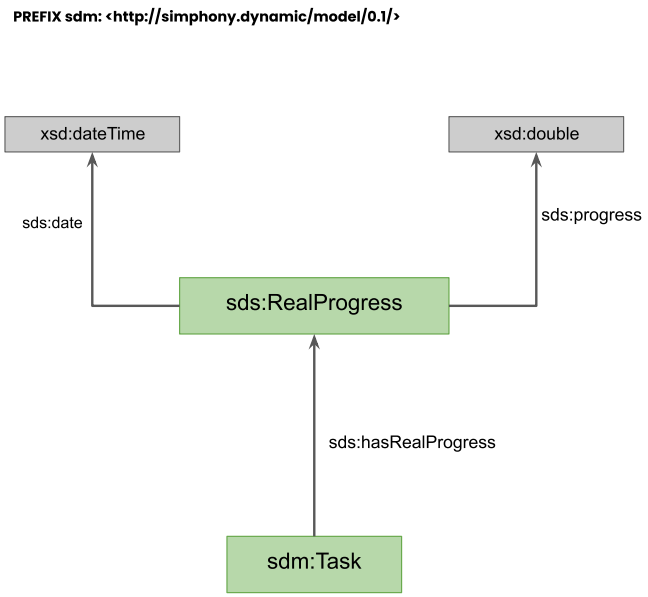


Figure 3.3: Linking *RealProgress* to *Task* in the project-specification model.

original model persisted in the database, enabling model reusability. This process happens automatically without any user intervention whenever the simulation is requested. Our method, however, does not use Bayesian Inference to update the productivity of the tasks, which would have made the updates more accurate.

3.4 Digital Construction Ontologies

To enhance the potential for interoperability of *Simphony Dynamic* , we looked into existing construction ontologies, and we identified the Digital Construction Ontologies (DiCo) [10] as the most extensive due to its coverage of all aspects of construction. DiCo facilitates semantic interoperability among different entities and procedures of the construction sector. Eight ontologies are included within DiCo to represent the different phases of the construction process: Contexts, Variables, Entities, Processes, Agents, Information, Materials, Occupancy, Lifecycle, and Energy.

More specifically, the Digital Construction Processes ontology was of particular interest to us as it is designed to describe the management of construction projects, essentially the same domain covered by the *Simphony Dynamic* project-

specification model. Figure 3.4 shows an overview of the Digital Construction Processes ontology.

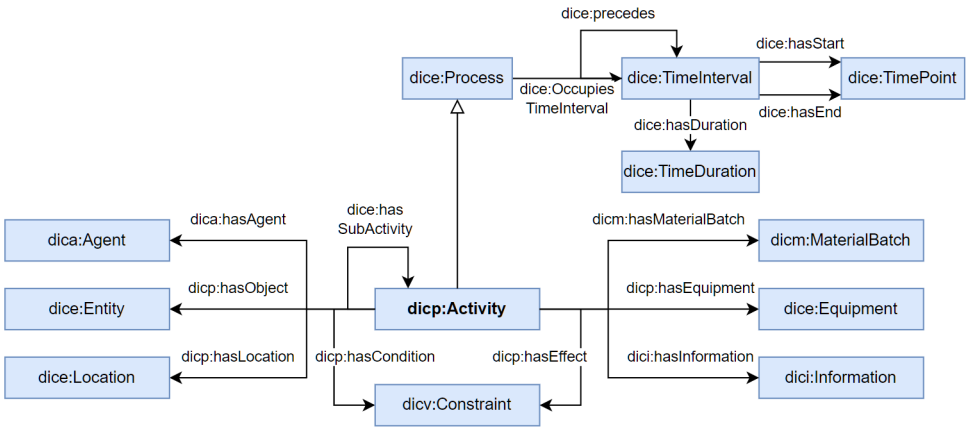


Figure 3.4: A diagram of the DiCo process ontology

The DiCo Processes ontology defines the DiCo concept Activity, as a process performed by actors or stakeholders. Activities are distinct, purposeful processes that have defined start and end times. Activities in DiCo can be basic, or complex and composed of other Activities, in effect resulting in a hierarchy of Activities.

All Activities in DiCo are performed by Agents, where an Agent is defined as a ”Person, organization, or organizational unit involved in a construction process”. An Activity or a collection of activities produces or is concerned about an Entity, which is a physical object or an occurrence. The MaterialBatch and Equipment concepts linked to an Activity describe what materials and equipment are needed to complete an activity.

3.5 The Aligned *Simphony Dynamic* -DiCo RDF Schema

Figure 3.5 shows how we have aligned DiCo with the *Simphony Dynamic* project-specification model.

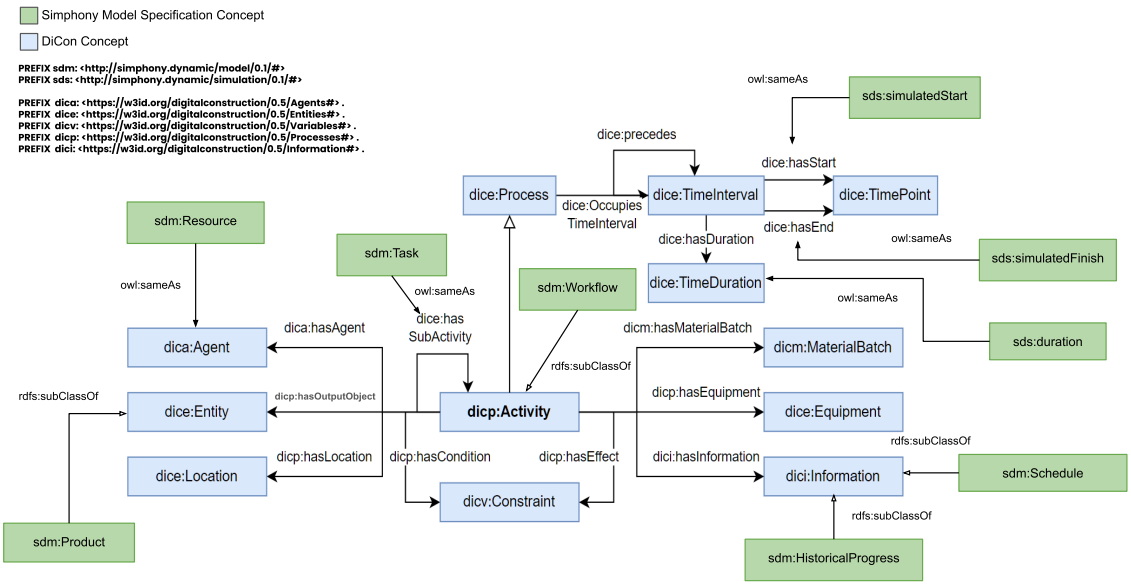


Figure 3.5: Alignment of the *Simphony Dynamic* project-Specification model the DiCo Process Ontology

At the core of the two models lie the concepts of *Activity* (DiCo) and *Task* (*Simphony Dynamic*) which correspond well to each other. As well both models capture the concept of compositionality of Activities/Tasks albeit in a different manner. DiCo allows for a recursive hierarchy of Activities composed of other Activities until at the end the composition elements are basic Activities. In *Simphony Dynamic* , *Workflow*s are *Task* compositions. Thus, in our aligned schema, *Workflow*s are considered equivalent to top-level Activities, and Tasks are considered equivalent to Basic Activities.

DiCo Activity is a subclass of DiCo Process, and a process has 3 properties -*hasStart*, *hasEnd*, and *hasDuration*. These 3 concepts can be directly mapped with *simulatedStart*, *simulatedFinish*, and *duration* of simulation results as the same concepts, as they all represent the start, end, and duration of an Activity or Task

All Activities in DiCo are performed by Agents, where an Agent is defined as a ”Person, organization, or organizational unit involved in a construction process”. *Simphony Dynamic Crews* are equivalent to *Agents*, as they are both conceived to be the organizational units that complete the Tasks/Activities.

Simphony Dynamic Products correspond to *Entities*: they are the physical objects constructed through *Workflow*s or the conceptual milestones accomplished once the *Workflow*s have been completed.

The *Simphony Dynamic* concepts of *Schedule* and *HistoricalProgress* do not have direct equivalents in DiCo but they can be viewed as *types of* DiCo concept *Information* associated with tasks to facilitate their simulation.

The purpose of this alignment is to make the data structure more understandable and interoperable to the people of construction. DiCo is used by the Building Information Modelling Management System (BIMMS) [26] platform which has created 6 tools that utilize this ontology. DiCo was evaluated in 5 different criteria: Clarity, Coverage, Consistency, Extendibility, and Usability. Although some of the evaluation criteria were subjective, DiCo being a high-level construction ontology leaves room for extensibility, which makes it flexible and favorable for creating a more domain-specific construction ontology or schema. By aligning our data model with DiCo, we can claim that it will also be of the same quality as DiCo in these 5 criteria.

Chapter 4

Simphony-as-a-Service: *Simphony*$_{aaS}$

4.1 The Migration Process

The Simphony Dynamic simulation environment is originally written on the .NET Framework 4.6, which is a Microsoft Windows-specific framework. However, .NET Framework 4.6 was retired by Microsoft on April 26, 2022, and support for it will end on January 12, 2027, which effectively makes Simphony a legacy software. Microsoft is developing a cross-platform development framework, called .NET which is currently in version 8.0. Simphony Dynamic is a Windows-only desktop application, but the simulation engine has the potential to be an online software service if repackaged into a Software-as-a-service model and it can be deployed in the cloud for better accessibility. To turn Simphony Dynamic into a SaaS (software-as-a-service) model deployed in the cloud, we wanted to port it to .NET 8.0 to make sure it could run both on Windows and Linux, so there are no OS constraints. Linux servers are much more popular than Windows servers because it is free of cost, and working with Linux servers is deemed easier due to larger community support. So it was very important for our service to run on Linux as well as Windows to allow our system to be versatile in terms of deployment options. To migrate Simphony Dynamic from .NET Framework 4.6 to .NET 8.0, we first looked into how we can wrap the Simphony Dynamic simulation engine using ASP.NET Core 8.0, a web framework that is based on .NET 8.0, to expose REST APIs to facilitate simulation by remote clients.

The first step of this migration process was to create a Nuget package of Sim-

phony Dynamic. A Nuget package is a dependency library/package in .NET that can be imported into a project. Firstly, we created an empty .NET Framework 4.8 Class Library project and moved the Simpony Dynamic code into it. Secondly, we removed the GUI (Graphical User Interface) code from this project because it had Windows-specific dependencies. Thirdly, we ran the project to generate the NuGet package.

The second step of the migration process was to create a Nuget package of Simphony Core Services, as it is a dependency for Simphony Dynamic. We used the same process as the previous steps, but as Simphony Core Services has no user interface, we did not have to deal with GUI code.

The third step was to import these two Nuget packages into our .NET 8.0 project and test if they could be imported and used.

The last step was to use ASP.NET Core 8[27] web framework to create a REST API that allows the user to request a simulation by sending a JSON representation of the simulation model and getting the response in a JSON format when the simulation finishes.

Migration from Microsoft Access DB to GraphDB Simphony Dynamic uses Microsoft AccessDB as its persistence storage. As we have identified in the previous Chapter, this hampers the interoperability and extensibility of the system significantly. The main reasons for this are - (1) Microsoft Access DB stores data in local files, so sharing any model or simulation result requires file sharing (2) If any updates are made to the model, then the local files have to be shared with everyone to get the updates (3) there is no database authentication or authorization so there is no data security layer present. We found these to be very big drawbacks, especially the security aspect. As a result, updating the database was essential.

Before migrating, we wanted to understand what type of database would work best with Simphony Dynamic. After analyzing the data structure and queries required, we came to the following conclusions

1. The Simphony Dynamic data structure can be represented using a graph. A model in *Simphony Dynamic* is a network of tasks represented using a Di-

rected Acyclic Graph. This led us to conclude that a graph database would be well-suited for our purpose

2. Any future analysis of model and simulation data, especially if it consists of graph traversal techniques, can be done easily using a graph database

3. We wanted to align the simulation modeling elements of Simphony Dynamic with a construction digital twin ontology, to do that a graph database would be ideal as it has terminologies and methods to represent inheritance and equivalent object classes.

We looked into popular graph database implementations and chose GraphDB [28]. The reasons for choosing GraphDB are - (1) support for all RDF file formats (2) Available as a cloud deployment on Microsoft Azure or AWS (3) better visualizations compared to competitors (4) built-in graph path search algorithms that can be used in SPARQL.

Adapting a microservices architecture After the migration, we looked into what architecture is best suited for our needs. We wanted our system to be deployable to the cloud, and as efficient and cost-effective as possible. Also, as we wanted our system to be a SaaS (software-as a service), scalability, and availability had to be taken into account. Considering these requirements, we concluded that the microservices architecture was best suited for our needs.

Each microservice will need to take care of only its responsibility, increasing efficiency. It is cost-effective to deploy and maintain smaller codebases in the cloud as the system will only scale the frequently used microservices if needed and not the whole codebase. Scalability is increased as microservices that are used more frequently can be scaled horizontally in the cloud to maintain performance. Multiple instances of the microservices that are crucial to the simulation process can be deployed to increase availability.

We conducted a feasibility study to find out if *Simphony Dynamic* can be broken into microservices, how many microservices can be created, and what would be their responsibilities. We identified two points of interest we could where the

	Simphony Core Services	Database Interactions
Responsibility	Discrete Event Simulation	Persistent Storage
Codebase Size	122238 Lines of code	1722 Lines of code
Point of coupling with *Simphony Dynamic*	Simulation execution	Read/Save Model and Simulation Result
Degree of coupling with *Simphony Dynamic*	High	Low
Migration effort	Extremely high	Low

Table 4.1: Factors considered for the feasibility study on adapting microservices architecture for $Simphony_{aaS}$.

project could be broken down into microservices - (1) The interface of Simphony Dynamic with Simphony Core services (2) The database reads and writes.

When we investigated the interfacing of Simphony Dynamic with Simphony Core Services, we found an extremely tight coupling within the code. For each simulation run, they communicate with each other very frequently. Hence, we concluded that they cannot be broken apart and should be kept together. If these two are broken into microservices, the network latency overhead of these many interactions will easily exceed the actual processing time.

The interactions with the database were found to be fairly limited. There were only 3 instances when the database was queried - (1) when loading the model (2) when saving the model after any changes (3) saving the results after the simulation was run. So, we decided to separate the code that dealt with the database into a microservice. Table 4.1 summarizes the result of the investigation.

We determined we could break Simphony Dynamic into 2 microservices:

1. **Simphony Simulation Microservice** - this microservice would contain Simphony Dynamic and Simphony Core Services as Nuget packages and would expose a REST API to allow remote clients to run simulations

2. **Repository Microserivce** - this microservice would interact with GraphDB to load and save models and simulation results. It would expose REST APIs for remote clients for this purpose.

4.2 Re-Architecting *Simphony Dynamic* into $Simphony_{aaS}$

$Simphony_{aaS}$ is based on a microservices architecture. The system consists of two

microservices and a suite of user interfaces. For project managers to easily interact with the Simphony simulation system, we have developed three different user interfaces catered to their needs: (i) a dashboard for monitoring project progress and simulating what-if scenarios, (ii) a model editing user interface to create a new simulation model or edit the baseline of an existing model, and (iii) a mobile application

The overall architecture is shown in Figure 4.1.

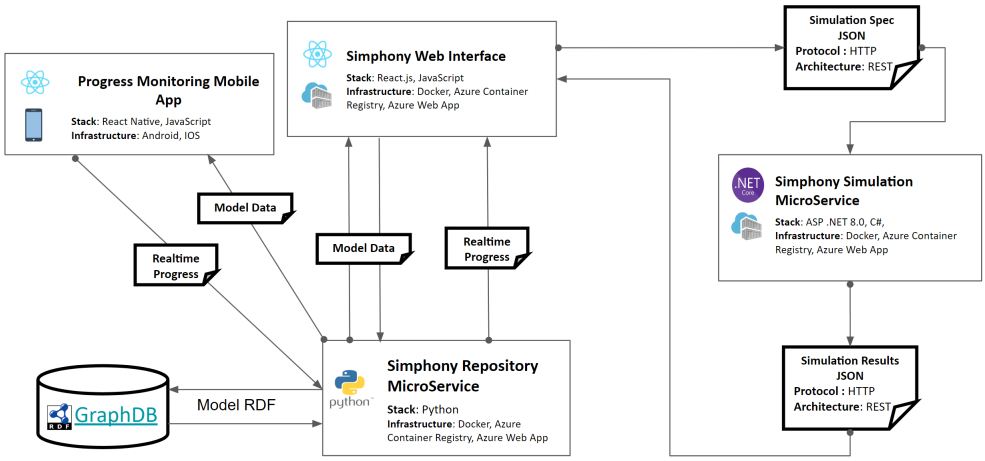


Figure 4.1: Architecture Diagram of *Simphony*$_{aaS}$

4.2.1 The Simulation Microservice

The Simulation Microservice is the heart of our system: it wraps the original *Simphony Dynamic* simulation engine and exposes a REST API, supporting the HTTP POST method, to facilitate simulation from remote clients. The API accepts a simulation model represented in the *Simphony Dynamic* Model specification in a JSON format. The internal architecture of this microservice is shown in Figure 4.2. Four steps occur when a simulation request is sent to the microservice:

1. The microservice accepts the JSON and validates the model, if the validation fails then the service returns an error message with a message pointing to where the error occurred.

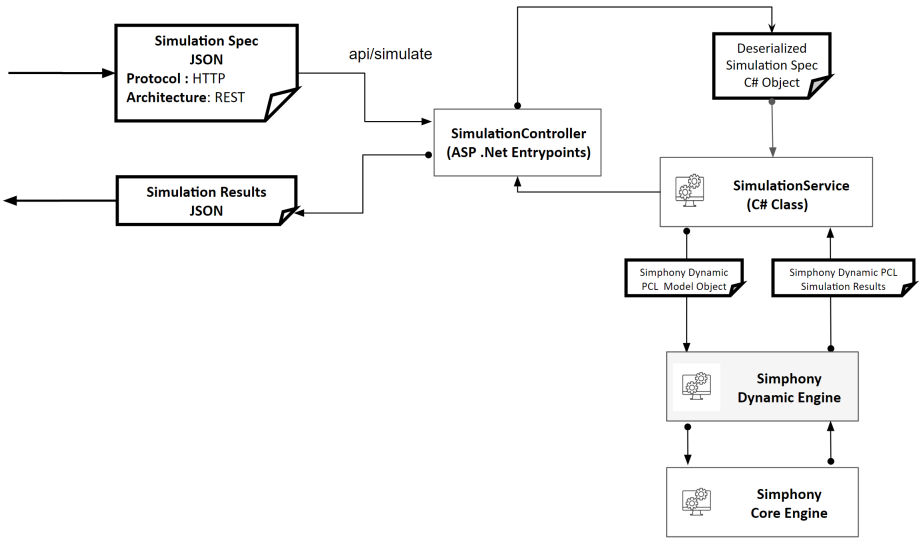


Figure 4.2: Internal architecture of the Simulation Microservice

2. If the JSON is valid, it is deserialized to a C# object instance which is understandable to the *Simphony Dynamic* simulation Engine.

3. The simulation engine accepts the model object instance as input and runs the simulation to return the result.

4. The simulation result is serialized into a JSON and returned to the user.

There is only one HTTP endpoint in this microservice to facilitate simulation [3].

4.2.2 The Repository Microservice

The Simphony Repository Microservice connects to the database and performs all required queries on the GraphDB repository. It contains all the SPARQL queries and exposes HTTP REST APIs to communicate with other microservices. Whenever a request is made to the service, a SPARQL query is executed on the database, and the result is serialized to a JSON and returned to the user. All kinds of interaction with the database must occur through this microservice. This ensures that read and write operations have strong consistency, and keeps the database secure as it is

[3]The API documentation of the endpoint is available at: https://shorturl.at/kuBH2

not being exposed to the web directly. The microservice is written in Python using the FastAPI web framework.

The GraphDB database contains all the simulation models and results and is the only persistent memory of our system. *Simphony Dynamic* uses Microsoft Access DB, which is a relational database to store data. Although a relational database works completely fine for reading and writing simulation data, a graph database has some significant advantages in our case as the data itself is represented as a graph of construction tasks. Especially when it comes to queries that require graph traversal, SPARQL queries can significantly outperform SQL queries. This is especially helpful for scenarios where data analysis is required within the task network. The next portion discusses some key questions that can be answered using SRAPQL and GraphDB.

Q. Which tasks require a particular crew e.g. Electrical Crew?

A crew lead might want to know which tasks their crew is assigned to. This information can help him create shift assignments for the crew.

```
PREFIX sds: <http://simphony.dynamic/simulation/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?taskname
WHERE{
    ?task sds:hasCrew ?crew .
    ?task rdfs:label ?taskname .
    ?crew rdfs:label "Electrical" .
}
```

Q. What is the predicted status i.e. percentage of work that should have been completed on a particular date, according to the baseline simulation data?

With the real data at hand, the project manager might want to compare the real project progress to the baseline progress.

```
PREFIX sdm: <http://simphony.dynamic/model/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
PREFIX sds: <http://simphony.dynamic/simulation/0.1/>
SELECT (SUM(?maxProgress)/180 as ?totalProgress) WHERE{
    SELECT ?product (MAX(?progress) as ?maxProgress)
    WHERE {
        ?progressData rdf:type sds:SimulationProgress .
        ?progressData sds:dateTime ?dateTime .
        BIND (xsd:dateTime(?dateTime) AS ?date)
        ?progressData sds:progress ?progress .
        ?progressData sds:productName ?product
        FILTER (?date < "2015-05-01T00:00:00"^^xsd:dateTime)
    }
    GROUP BY ?product
}
```

Q. Which products were completed on a certain date according to the baseline simulation?

Having the list of finished products from on-site data, the project manager might want to do a comparison of that list with the baseline. By doing this comparison, the project manager can know if the project is on schedule, or is ahead of schedule on a product level. The following query allows the project manager to get a list of products that were supposed to be completed according to the baseline.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sds: <http://simphony.dynamic/simulation/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?product ?productProgress WHERE {
{SELECT ?product (MAX(?progress) as ?productProgress)
    WHERE {
        ?progressData rdf:type sds:SimulationProgress .
        ?progressData sds:dateTime ?dateTime .
        BIND (xsd:dateTime(?dateTime) AS ?date)
        ?progressData sds:progress ?progress .
        ?progressData sds:productName ?product
        FILTER (?date < "2015-04-29T00:00:00"^^xsd:dateTime)
    }
        GROUP BY ?product }
    FILTER (?productProgress = 1.0)
}
```

Q. What tasks should be completed to start working on Task B, if the project

is currently on Task A?

If the project is currently at Task A and the stakeholders want to accelerate the work to start working on Task B, all the tasks that connect the path from Task A to Task B should be completed.

This question is particularly interesting as it uses the capabilities of graph traversal built into GraphDB, which is difficult to query in a relational database like MS AccessDB.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX path: <http://www.ontotext.com/path#>
PREFIX sdm: <http://simphony.dynamic/model/0.1/>
PREFIX Task: <http://simphony.dynamic/model/0.1/Task/>

SELECT ?start ?end ?index ?startName ?endName
WHERE {
    VALUES (?src ?dst) {
        ( Task:A Task:B )
    }
    SERVICE <http://www.ontotext.com/path#search> {
       <urn:path> path:findPath path:allPath ;
       path:sourceNode ?src ;
       path:destinationNode ?dst ;
       path:resultBinding ?edge ;
       path:resultBindingIndex ?index ;
       path:startNode ?start;
       path:endNode ?end;
       path:exportBinding ?startName;
       path:exportBinding ?endName .
       SERVICE <urn:path> {
           ?start sdm:hasSuccessorTask ?end .
           ?start rdfs:label ?startName .
           ?end rdfs:label ?endName .
        }
    }

}
}
```

Q. Is there a cycle starting from Task A?

The *Simphony Dynamic* Project-specification model is a essentially a Directed Acyclic Graph of tasks. If there is a cycle within the task network, the model is invalid. The

following query checks the validity of the model.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX path: <http://www.ontotext.com/path#>
PREFIX sdm: <http://simphony.dynamic/model/0.1/>
PREFIX Task: <http://simphony.dynamic/model/0.1/Task/>

SELECT *
WHERE {
    VALUES (?src) {
        (Task:A)
    }
    SERVICE <http://www.ontotext.com/path#search> {
        <urn:path> path:findPath path:cycle ;
                   path:sourceNode ?src ;
                   path:resultBinding ?edge ;
                   path:pathIndex ?path ;
                   path:resultBindingIndex ?index ;
                   path:startNode ?start;
                   path:endNode ?end;
        SERVICE <urn:path> {
            ?start sdm:hasSuccessorTask ?end
        }
    }
}
```

4.2.3 The Project-Specification Editor

The *Simphony*$_{aaS}$ Model Editor is a web-based model editing interface. The pre-existing simulation modeling GUI, which was created in .Net Framework 4.6 for *Simphony Dynamic* was redeveloped into a web interface using React.js. The editing interface presents a drag-and-drop system to create an activity network similar to *Simphony Dynamic* . The resources and their availabilities, the crew structure, and products can also be created and edited from this interface using tabular inputs similar to *Simphony Dynamic* . The user can then simulate the model and save the results in the database as the baseline schedule.

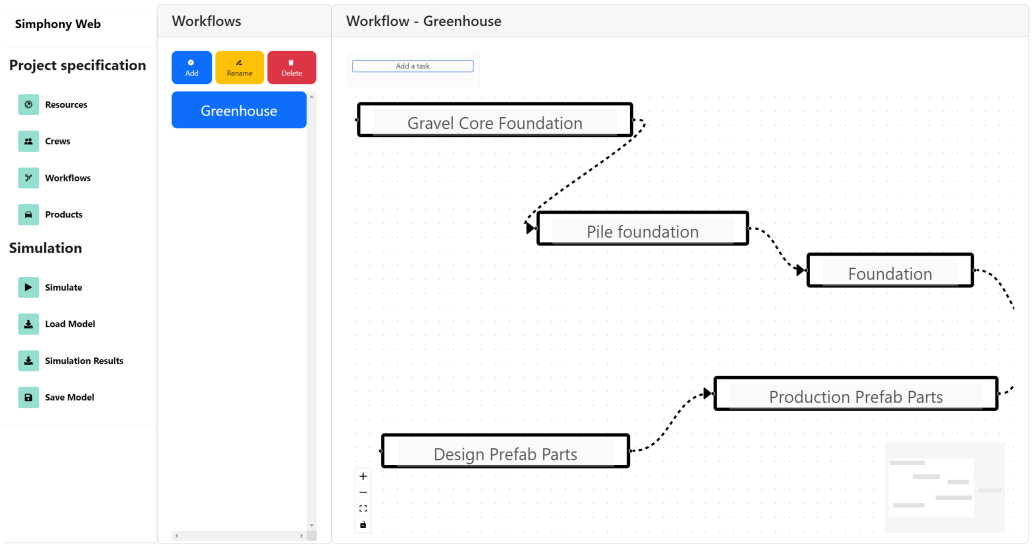


Figure 4.3: *Simphony*$_{aaS}$ Project-Specification Editor

4.2.4 The Project-Management Interface

The *Simphony*$_{aaS}$ Project-Management Interface is the primary user interface of our system where the user can track a project and simulate what-if scenarios. The interface consists of charts and tables for the user to view the project data and various comparative reports. Data from 3 different contexts are present in the Project-Management Interface:

- Baseline data - The result of the simulation that was run at the beginning of the project, to find the expected end of the project, and expected progress over time
- Present or Real data - Task progress data collected from the site as the project progresses in real life.
- Future data - The result of the new simulation that was run from the present date (or date selected from the date picker) to see what will happen in the future.

A snapshot of the Project-Management Interface is shown in Figure 4.4. At the left side of the user interface is a sidebar, with navigation links to the 6 different pages of the dashboard. At the top left, there is a button labeled **RELOAD DATA**.

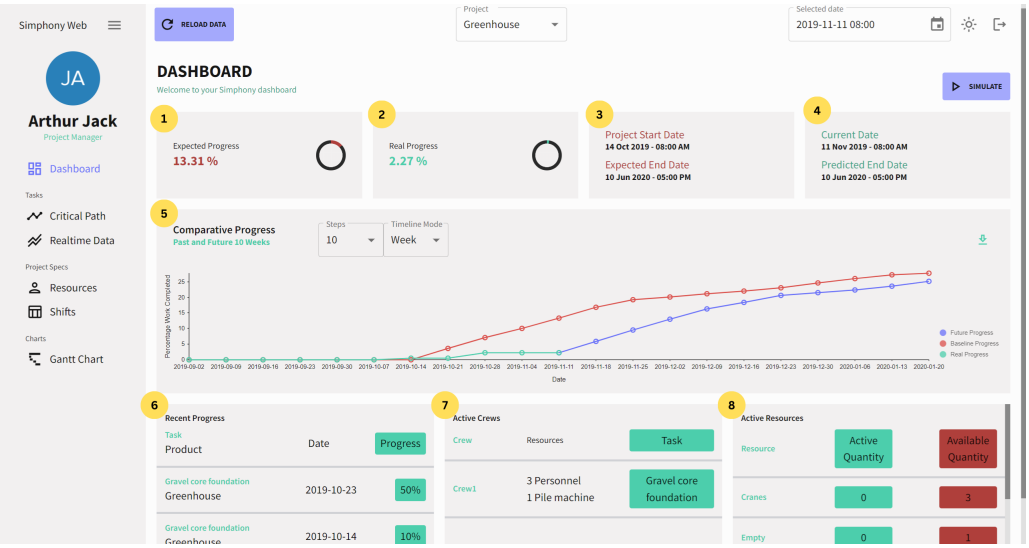


Figure 4.4: The dashboard page of the *Simphony*$_{aaS}$ Project-Management Interface

Clicking this button fetches the project specifications and real progress update data from the Repository microservice, and updates the user interface. At the top middle, there is a dropdown to change the selected project. At the top right, there is a date selector. This allows the user to navigate through the project data over time by selecting any date. The user interface is updated to view data based on the selected data. To the right of the date selector, there is a button to change the theme of the user interface between dark and light themes, and the rightmost button is a logout button.

There are six pages available in the Project-Management Interface:

The Dashboard page Figure 4.4 shows the Dashboard page of the management interface. The dashboard page is divided into 7 cards showing information from the past, present, and future of the project. The dashboard page also has a button, labeled **SIMULATE**, which can be pressed to run a new simulation from the selected date. The yellow circles with numbers have been added to the figure to refer to the user interface elements in the text. On this page, anything in red color indicates data from the baseline simulation i.e. expectations, green color indicates real progress and purple color indicates future predictions from simulation.

The top left card (marked as 1) shows the expected progress of the project on

the selected date. The next card on the right (marked as 2) shows the real progress of the project. The next card (marked as 3) shows the start date of the project and the expected end date, according to the baseline. The next card (marked as 4) shows the currently selected date, and when the simulation predicts the project will end, based on the current progress.

In the middle of the dashboard is a large card (marked as 5) that shows a comparative line chart between the baseline project progress (shown in red), real project progress(shown in green), and simulated future project progress (shown in purple). By default, the future and past 10 weeks are shown in the chart but can be configured using dropdown selectors on the card.

The bottom left card (marked as 6) shows the recent real progress of tasks, sorted by date in descending order. The next card to the right (marked as 7) shows which crews are currently active, and their resource allocations. The last card of the bottom (8) shows the resources that are currently active in the project, marked with green, and how many of them are available for work, marked with red.

Overall, the dashboard shows a summary of the past, present, and possible future of the project to the project manager.

The Critical Path page The Critical Path page shows the baseline and current critical path of the project. As the project progresses and tasks are completed, the critical path of the project can change. If the work is not adjusted to focus on the new critical path, then it will lag behind the expected deadline. The page shows this comparison in a tabular format. Figure 4.5 shows the Critical Path page of the Project Management Interface.

There are 6 columns in the table on this page. The first two columns indicate which Product and Task the data belongs to, working as a primary key for the data as these two together form a unique name of a task.

The next column named *Simulated Start* shows when the future simulation started this task. If the task is already done, the data is empty as a task already completed is not repeated in the future. The *Simulated Finish* column shows when the future simulation started and finished this task and is kept empty if the task has

already been completed.

The next two columns, *Baseline Critical Path* and *New Critical Path* show the comparison between critical paths of the task. If a task was part of the critical path in the baseline simulation, it is indicated with a *True* value and *False* otherwise in the *Baseline Critical Path* column. If a task is currently in the critical path, according to a new simulation (run using the SIMULATE button on the dashboard) the value on the *New Critical Path* column will be True, and False otherwise. The user can identify any discrepancies within these two columns to discover changes in the critical path.

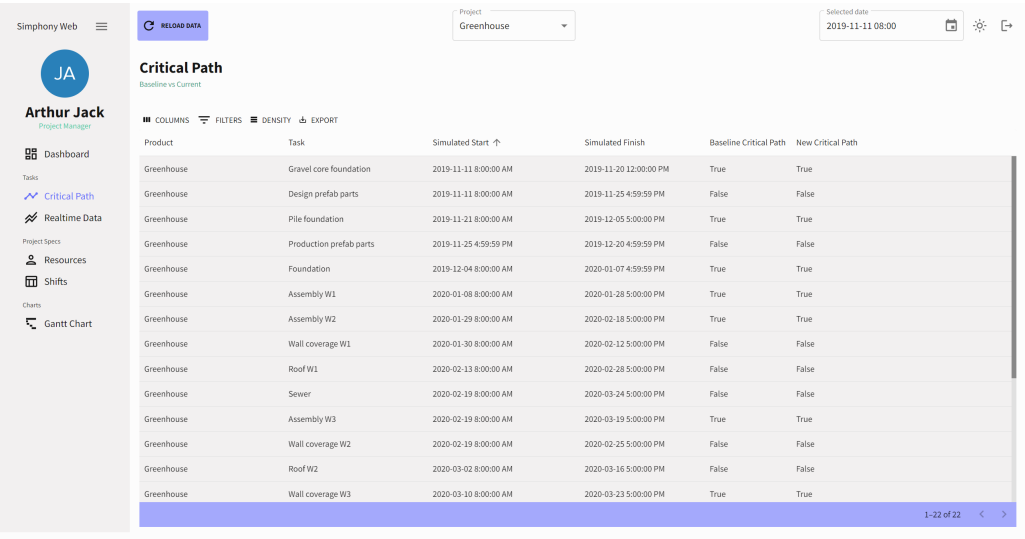


Figure 4.5: The Critical Path page of the $Simphony_{aaS}$ Project-Management Interface

The Real-time Data page The Realtime Data page shows the user the real on-site progress and how it compares to simulated progress. The data is presented in a tabular format and any of the columns can be filtered to find the desired task data. The user can look into each task individually and detect which tasks need more attention by comparing the man hours completed of the tasks. A snapshot of the real-time data page is shown in Figure 4.6.

There are 6 different columns on this page. The *Product* ad *Task* column indicates which task and which product the data is related to.

The *Budgeted Man Hours* column shows how many man hours were initially budgeted or estimated for this task. The *Simulated Earned Man Hours* column specifies how many man hours were completed on the selected date, according to the baseline simulation. The *Earned Man Hours* column shows how many man hours have been completed in real life and the *Percentage Complete* column shows real progress in percentage.

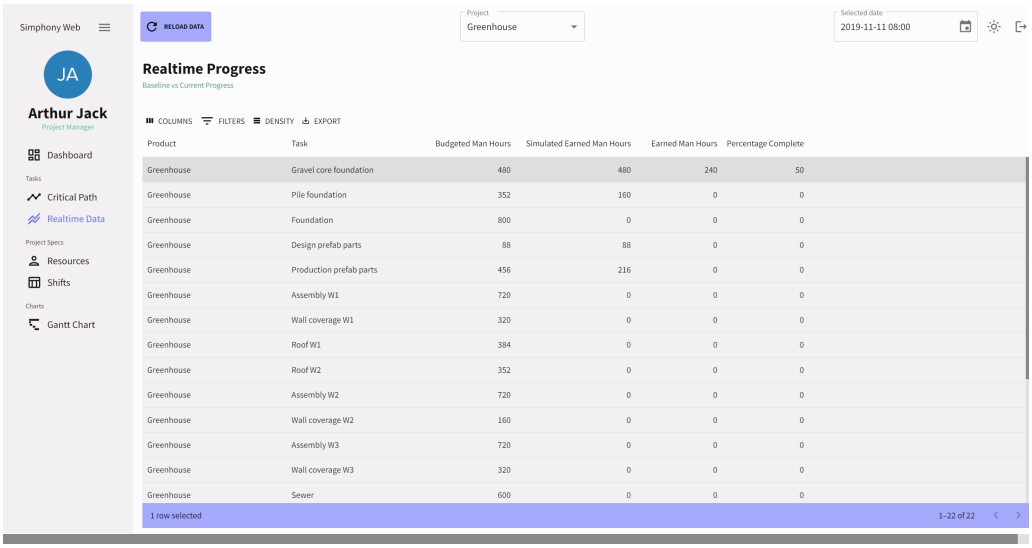


Figure 4.6: The Realtime Data page of the *Simphony*$_{aaS}$ Project-Management Interface

The Resources page The resources page allows the user to edit the resource configuration of the current project to explore what-if scenarios. Figure 4.7 presents a snapshot of the Resources page. At the top of the page is a dropdown, where the selected resource can be changed. In the figure, we can see that the ”Cranes” resource is selected. The table has a toolbar with options for filtering columns and a button to add new records. Using the *Actions* column the user can edit or delete an entry. The *Available From* and *Quantity* columns indicate what quantity of resources are available and from what date. A project manager can change the resource configuration of his project from this page, and re-simulate the project by going back to the dashboard page and pressing the **SIMULATE** button.

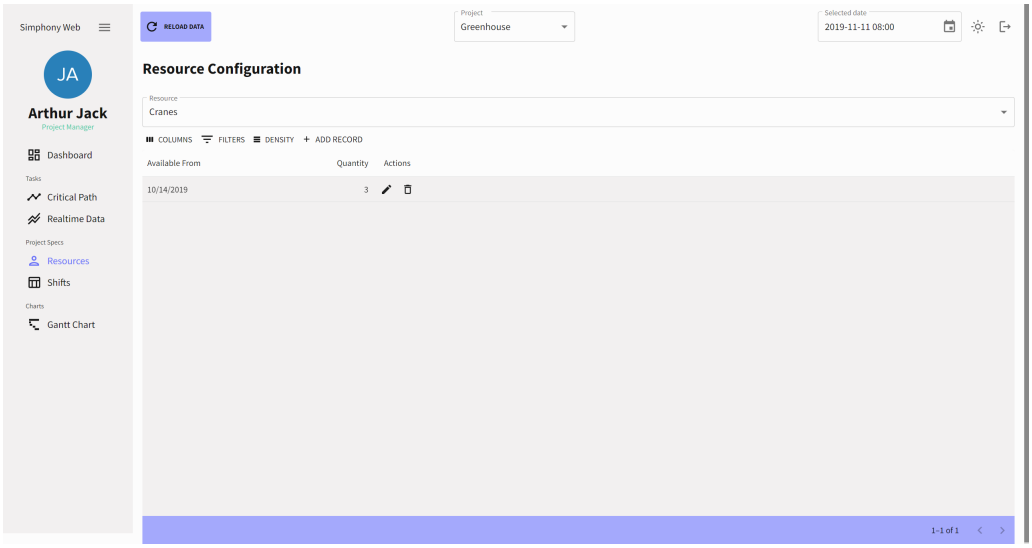


Figure 4.7: The Resources page of the *Simphony*$_{aaS}$ Project-Management Interface

The Shifts page The shifts page allows the user to edit the shift configuration of the current project. Figure 4.8 shows the shifts page. The user can select any day of the week to add, remove, or delete shifts from the dropdown at the top of the page. They can then add a new shift using the **Add Record** button on the toolbar at the top of the table or edit the shift using the action buttons on each record. After the user makes their changes, they can go back to the dashboard page and press the **SIMULATE** button to see how the new configuration changes future progress.

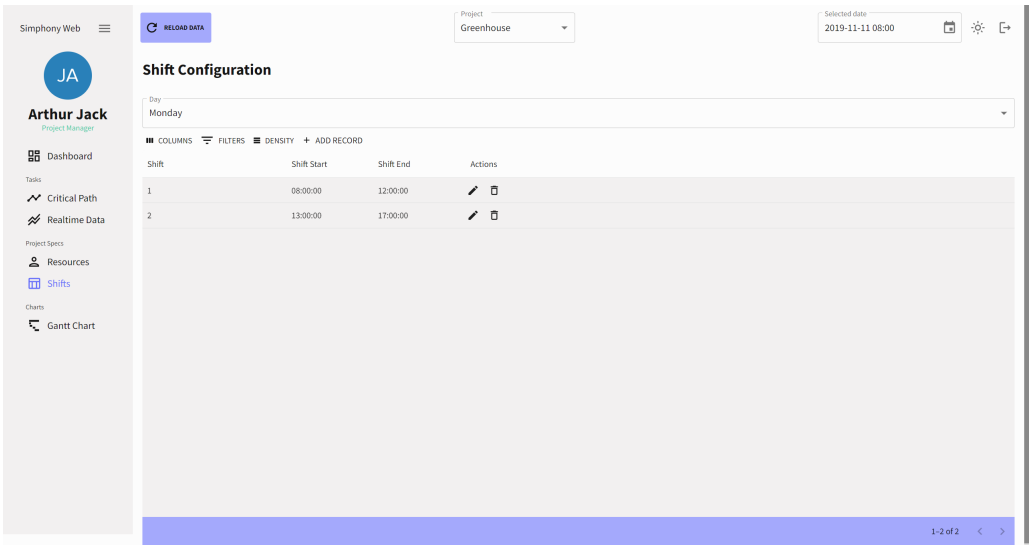


Figure 4.8: The Shifts page of the *Simphony*$_{aaS}$ Project-Management Interface

The Gantt Chart page The Gantt Chart page shows the baseline, current progress, and simulation results in a Gantt chart, as shown in Figure 4.9. The red color indicates baseline tasks, green indicates real progress, and purple indicates the predicted start and end times of the task according to the new simulation. It presents the results in both tabular and visual format and the views can be resized using drag and drop to view more of the visual part or the tabular part. The Gantt chart also shows the dependencies between the tasks using directional arrows. The timeline granularity (the horizontal granularity) of the Gantt chart can also be adjusted through a dropdown at the top of the page to properly fit the data on the screen.

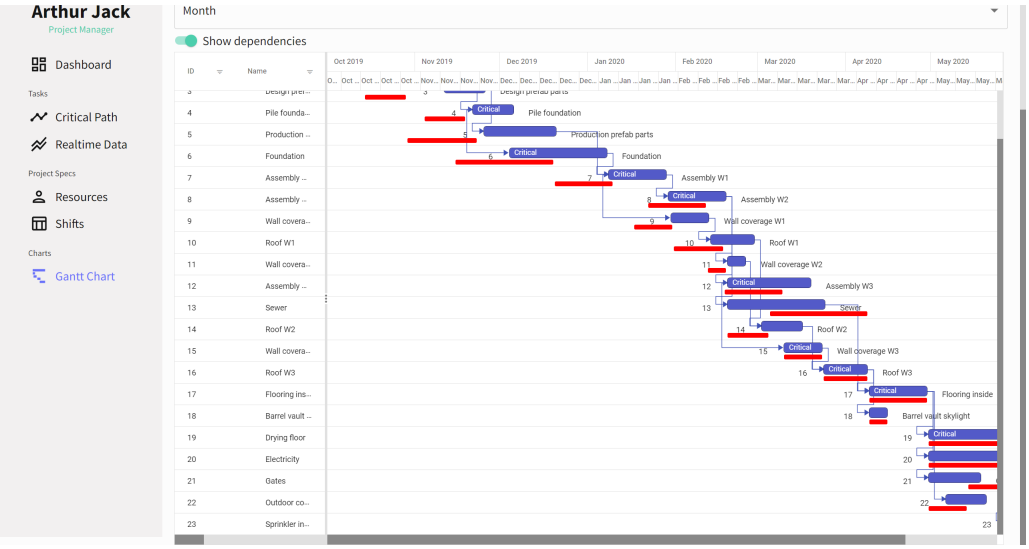


Figure 4.9: The Gantt Chart page of the *Simphony*$_{aaS}$ Project-Management Interface

4.2.5 The Progress-Monitoring Mobile Application

The Simphony Mobile App allows the user on-site (Construction Site Manager) to report the progress of work every day to the system after the end of the day. The application is very simple with only two input fields: which date the update corresponds to and what percentage of work of a task has been completed on that date. The user can select a particular date, search for an activity using its name, and enter the percentage of work done for that activity for that day through the application. The application connects with the Repository microservice to send daily updates to be stored in the database. The on-site crew leads can use the application to report

everyday progress. The Project Manager can view this update in the dashboard and can use this data for decision support.

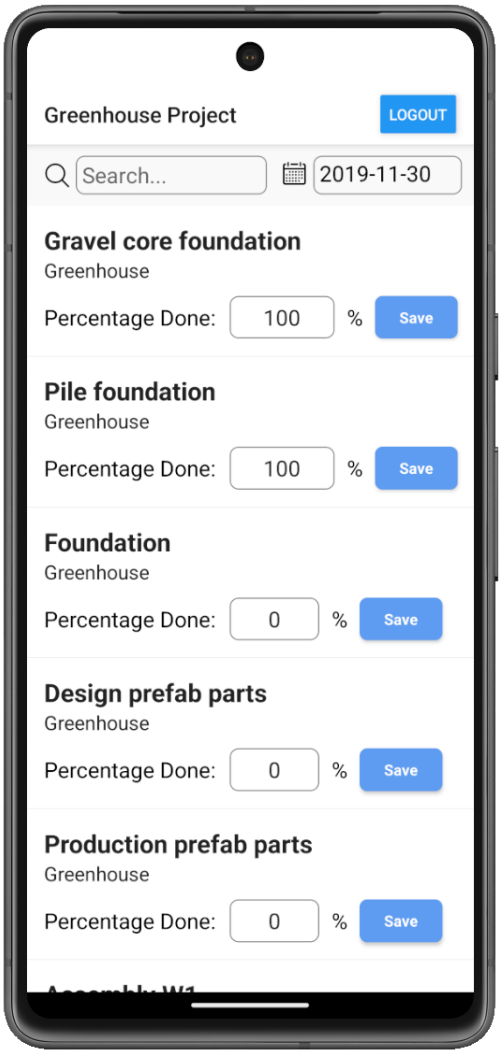


Figure 4.10: The user interface of *Simphony*$_{aaS}$ Progress-Monitoring Mobile Application

Chapter 5

An Illustrative Example

In this section, we will go through an example and show how our system provides decision support for project management. We will demonstrate the process of simulating a Greenhouse building model, progress tracking, model re-simulation from a selected point in time of the project, and explore what-if scenarios.

5.1 The Example Project

To demonstrate our the capabilities of $Simphony_{aaS}$, we chose a real construction project from a database of real project data collected by Batselier et al. [29], available publicly on their website [30]. We selected a project based on 4 criteria: (1) an activity count of 25 or less, to make the example simple to understand, (2) an industrial construction project, so that the terminologies are easily understandable, (3) a serial/parallel task ratio of more than 40% to demonstrate how our system handles parallel tasks (i.e. changes in the critical path) and (4) comprehensible task names for better understanding of the example. Based on these criteria, only 2 projects from the database were found eligible - the Greenhouse project (Project code 2019-08) and the Castelein project (Project code 2018-03)[4]. But within these 2, the Castelein project had incomplete resource allocation data so we chose the Greenhouse project.

[4]The project codes are assigned by the authors to all the projects in the database as unique identifiers.

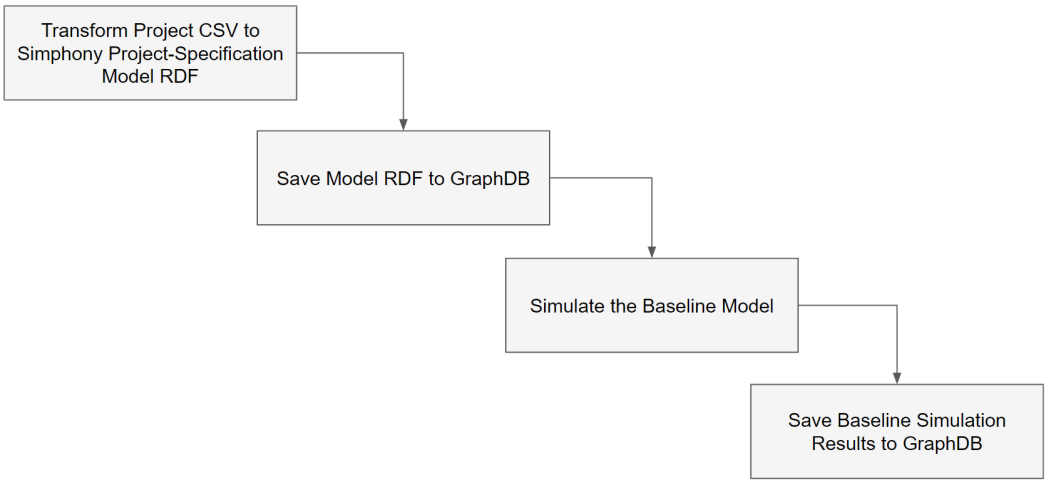


Figure 5.1: Transforming the Greenhouse project Excel file to Simphony Simulation-Specification model

5.2 Transforming the Example Data to *Simphony Dynamic* Specification Model

The Greenhouse project has 23 tasks, as shown in Table 5.1. The dependencies of each task, their baseline schedule, duration, and resource allocation are also described in the table. The data is available as a Microsoft Excel file in xlsx file format. We transformed the data into a Simphony Simulation-Specification model using a Python script. Figure 5.1 shows the process through which the script transforms the data. It is to be noted that the task *Finish* is not included in the model, as it only indicates the time of finish of the project and has no resources attached to it. So in the simulation model, we have 22 tasks.

However, the simulation model can also be created using the Model Editor Interface in a similar fashion to how it is done in *Simphony Dynamic* . At this point, the initial project specifications were ready with a baseline schedule, resource specifications, a task network, and baseline simulation results. The simulation results were the same as the planned baseline of the Greenhouse project, and it predicts that the project will end on 10 June 2020.

ID	Name	Predecessors	Successors	Baseline Start	Baseline End	Duration	Resource Demand
1	Gravel core foundation		FS2	14-10-2019 8:00	01-11-2019 17:00	15d	Personnel[3.00];Pile machine
2	Pile foundation	1FS	SS3+1w 2d	04-11-2019 8:00	18-11-2019 17:00	11d	Personnel[3.00];Pile machine
3	Foundation	2SS+1w 2d	FS6	13-11-2019 8:00	17-12-2019 17:00	25d	Personnel[3.00];Excavator
4	Design prefab parts		FS5	14-10-2019 8:00	28-10-2019 17:00	11d	Ext. personnel
5	Production prefab parts	4FS	FS6	29-10-2019 8:00	22-11-2019 17:00	19d	Ext. personnel[3.00]
6	Assembly W1	5FS;3FS	SS7+2w 2d;FS10	18-12-2019 8:00	21-01-2020 17:00	15d	Personnel[3.00];Cranes[2.00];Scissor Lift
7	Wall coverage W1	6SS+2w 2d	FS8	17-01-2020 8:00	30-01-2020 17:00	10d	Personnel[2.00];Cranes;Scissor Lift
8	Roof W1	7FS	FS9	31-01-2020 8:00	17-02-2020 17:00	12d	Personnel[3.00];Scissor Lift
9	Roof W2	8FS;11FS	FS15	19-02-2020 8:00	04-03-2020 17:00	11d	Personnel[3.00];Scissor Lift
10	Assembly W2	6FS	FS11;FS12;FS14	22-01-2020 8:00	11-02-2020 17:00	15d	Personnel[3.00];Cranes[2.00];Scissor Lift
11	Wall coverage W2	10FS	FS9	12-02-2020 8:00	18-02-2020 17:00	5d	Personnel[2.00];Cranes;Scissor Lift
12	Assembly W3	10FS	SS13+2w	18-02-2020 8:00	09-03-2020 17:00	15d	Personnel[3.00];Cranes[2.00];Scissor Lift
13	Wall coverage W3	12SS+2w	FS15	10-03-2020 8:00	23-03-2020 17:00	10d	Personnel[2.00];Cranes;Scissor Lift
14	Sewer	10FS	FS16	05-03-2020 8:00	08-04-2020 17:00	25d	Personnel[2.00];Excavator
15	Roof W3	9FS;13FS	FS16;FS22	24-03-2020 8:00	08-04-2020 17:00	12d	Personnel[3.00];Scissor Lift
16	Flooring inside	15FS;14FS	FS17;FS19;FS20;FS21	09-04-2020 8:00	29-04-2020 17:00	15d	Ext. personnel[25.00]
17	Drying floor	16FS	FS18	30-04-2020 8:00	27-05-2020 17:00	20d	
18	Sprinkler installation	17FS	FS23	28-05-2020 8:00	10-06-2020 17:00	10d	Personnel[2.00];Scissor Lift
19	Electricity	16FS	FS23	30-04-2020 8:00	09-06-2020 17:00	29d	Personnel[4.00;Scissor Lift
20	Gates	16FS	FS23	14-05-2020 8:00	27-05-2020 17:00	10d	Personnel[2.00;Scissor Lift
21	Outdoor construction	16FS	FS23	30-04-2020 8:00	13-05-2020 17:00	10d	Personnel[2.00]
22	Barrel vault skylight	15FS	FS23	09-04-2020 8:00	15-04-2020 17:00	5d	Personnel[3.00];Scissor Lift
23	Finish	18FS;19FS;20FS;21FS;22FS		10-06-2020 17:00	10-06-2020 17:00	0	

Table 5.1: Tasks, dependencies, schedule and resource allocation of the Greenhouse project

5.3 The Project Management Dashboard

The project starts on October 14, 2019. We assume a scenario where work is done every day and through the mobile application, the site manager reports the daily progress. We also assume that until 10 December 2019, the first 2 tasks from table 5.1, *Gravel core foundation*, and *Pile foundation* were completed. We can see in Figure 5.2 that for the Greenhouse project, 21.27 % of the project was expected to be finished on 10 December 2019, but 9.09% was finished. The expected end date for the project is 10 Jun 2020 at 5:00 PM. The current selected date is 10 December 2019, and the predicted end date is not available as the user has not run a simulation yet.

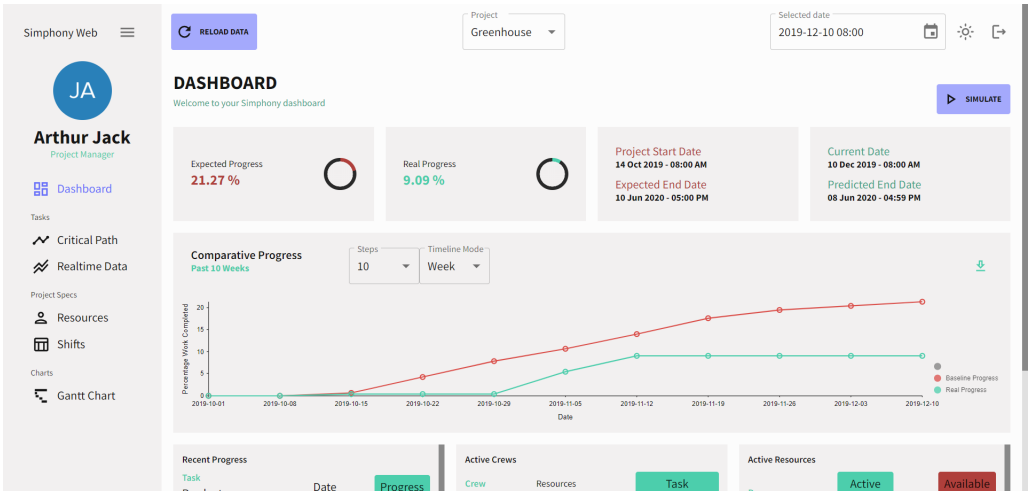


Figure 5.2: The dashboard page of the Project Management Interface for the Greenhouse project on 10 December 2019, before simulation.

Once the user presses the simulation button and the simulation is complete, the dashboard is updated with the simulation results. The new simulation predicts the future from 10 December 2019, so we refer to it as future data. Figure 5.3 shows the dashboard after it is populated with the real data. We can see the simulation predicts that the project will end on 24 June 2020, pushing back the project by 14 days. The line chart shows the comparative analysis of the project's progress. The green line shows the real progress of the project up to 10 December 2019, and the purple line continues this line to show how the project will progress in the future. The red line indicates the baseline progress over time. For the current scenario, we

observe that the baseline progress line is always above the future prediction for the next 10 weeks. So, the project is likely to be behind for the next 10 weeks.

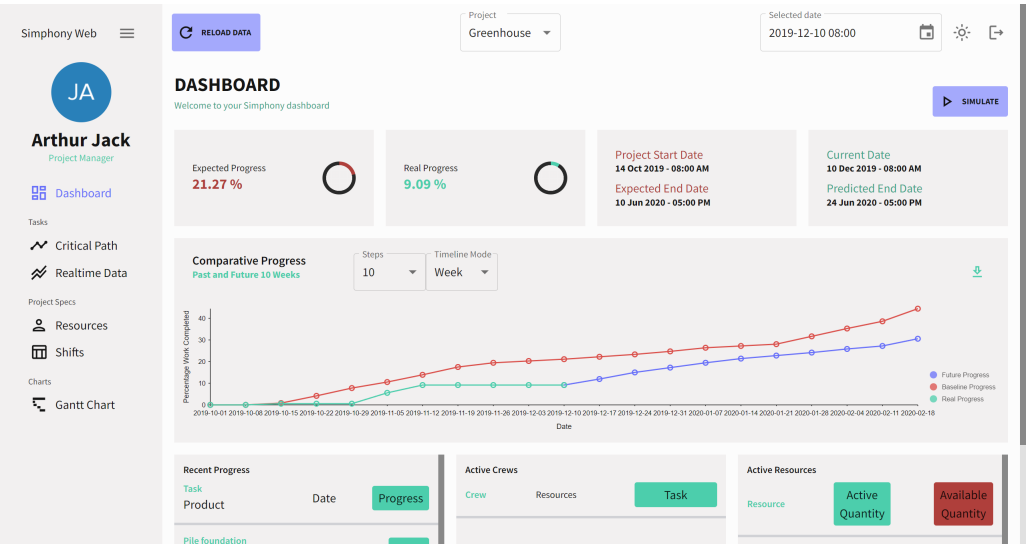


Figure 5.3: The dashboard page of the Project Management Interface for the Greenhouse project on 10 December 2019, after simulation.

5.4 Comparative Critical Path Analysis

At the beginning of the project, the critical path for the Greenhouse project was: *Gravel Core Foundation →Pile Foundation →Foundation →Assembly W1 →Assembly W2 →Assembly W3 →Wall Coverage W3 →Roof W3 →Flooring Inside →Drying floors →Sprinkler installation.*

But after the first two tasks in the critical path have been completed, it changes to: *Design prefab parts →Production prefab parts →Foundation →Assembly W1 →Assembly W2 →Assembly W3 →Wall Coverage W3 →Roof W3 →Flooring Inside →Drying floors →Sprinkler installation.*

In Figure 5.4, we can see how this change is shown in the Critical Path page of the interface. The New Critical Path column shows the critical path calculated based on the real-time progress on the selected date. This was calculated when the SIMULATE button was pressed on the dashboard. The Baseline Critical Path shows which tasks were in the critical path at the beginning of the project. The columns Simulated Start and Simulated Finish are the results of the simulation, showing

when the task is expected to start and finish. The *Design prefab parts* and *Product prefab parts* are now on the critical path, but were not on the baseline critical path.

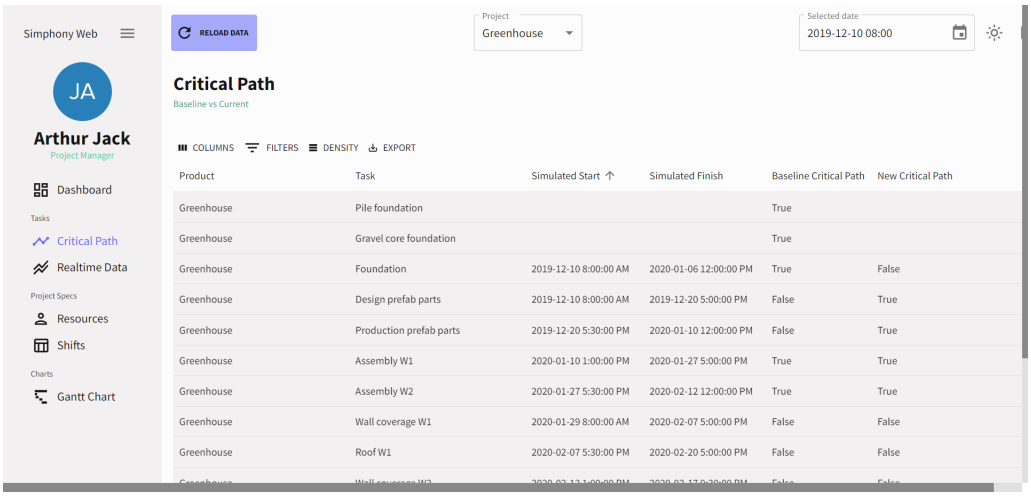


Figure 5.4: The Critical Path page of the Project Management Interface for the Greenhouse project on 10 December 2019

5.5 Tracking Real Progress and Comparison with Baseline

The Realtime Progress page, as shown in Figure 5.5 presents a table with the real progress of all the activities, reported using the Progress-Monitoring Mobile Application. As the crew leads report the progress of each task every day, this panel updates the data for the project manager to view. Through this single table view, the project manager can compare how much work has been done and compare it to the baseline simulation and budgeted man hours. In Figure 5.5, we can see that the first 2 tasks have been fully completed by 10 December 2019. According to the baseline simulation, the tasks *Design prefab parts* and *Production prefab parts* should have been completed by this time, but the real data shows these tasks have not yet been started. So, the project manager can quickly decide to put more focus on these tasks to accelerate the project. This panel is one of the most important parts of the Project-Management Interface as it provides a detailed expectation vs reality overview of the project.

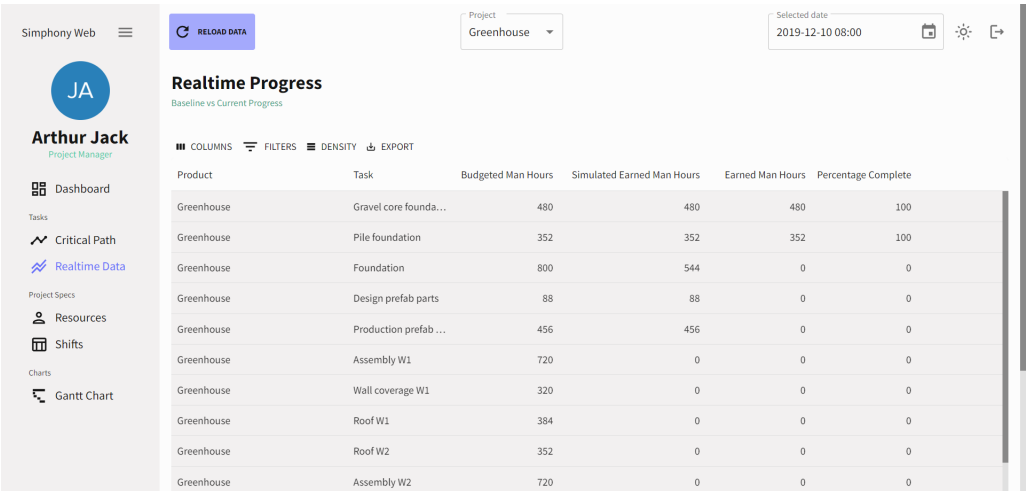


Figure 5.5: The Realtime Progress page of the Project Management Interface for the Greenhouse project on 10 December 2019

5.6 The Gantt Chart View

The Gantt Chart panel shows the simulation results, the baseline schedule, and also the progress of each task. The red color shows the baseline schedule, the purple color shows the future simulation results and the green color shows the current progress. In Figure 5.6, we can see that the tasks *Gravel core foundation*, and *Pile foundation*, have been fully completed as they are fully green, indicating their progress is 100%. The other tasks have not been started yet so their start and end dates are simulated, indicated by their purple color. The dependencies between the tasks are shown using directional arrows. The tasks that are on the current critical path and labeled ”Critical” on the Gantt Chart. The left side of the Gantt Chart presents these data in a tabular way, as shown in Figure 5.7

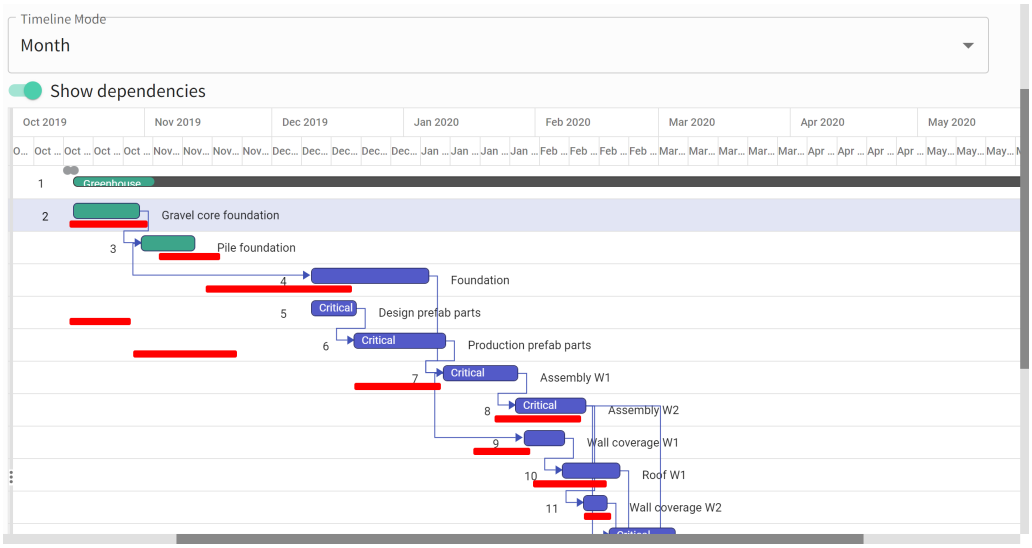


Figure 5.6: The Gantt Chart page of the Project Management Interface for the Greenhouse project on 10 December 2019

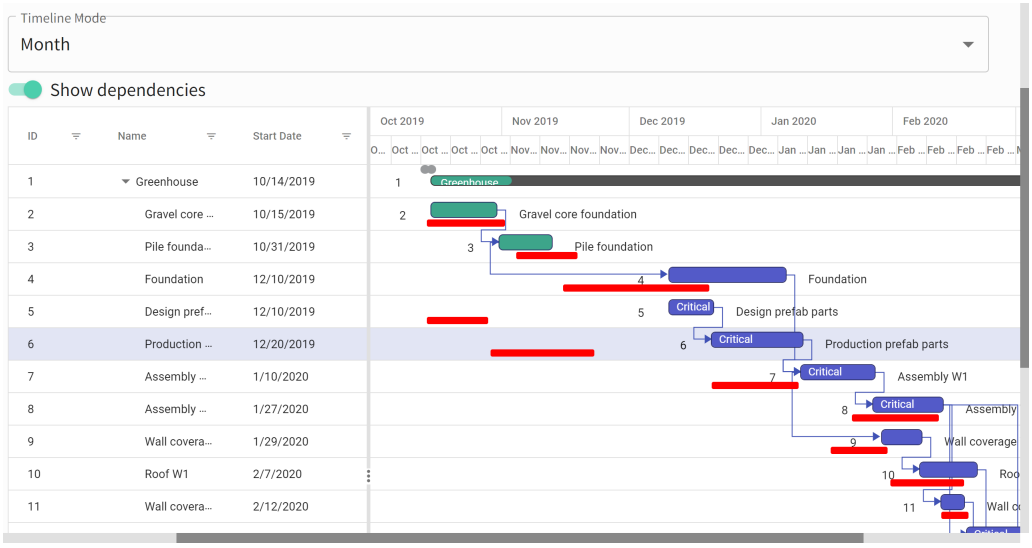


Figure 5.7: The tabular view of the Gantt Chart page of the Project Management Interface for the Greenhouse project on 10 December 2019

5.7 Simulating What-If Scenarios

As we have seen in our example scenario, the Greenhouse project is behind schedule on 10 December 2019 and is expected to finish on 24 June 2020 instead of 10 June 2020, a delay of 14 days. To decide what can be done to accelerate the project, what-if scenarios can be simulated through the Project Management Interface by

changing the number of resources allocated or by editing the work time i.e. adding or removing more shifts.

The project manager for the Greenhouse project might decide to hire more people to accelerate the project, but before doing so they can test if it will help to meet the deadline from the Resources panel of the Project Management Interface. The user can select which resource they want to allocate more of and add or edit the availability for that resource. Let us assume a scenario where the project manager wants to hire 1 more crane and 6 more personnel on 10 December 2019 and see how it changes the project timeline. Figure 5.8 shows the resource configuration after the changes are made by the project manager in the interface. After re-simulating the project using the new resource allocations, our system shows that the project is projected to end on 23 July 2020. So, adding more resources will accelerate the project by 1 day.

Another what-if scenario the project manager might want to look into is adding more work hours. For the Greenhouse project, let us assume the project manager decides to add an extra 2-hour shift on Mondays, Wednesdays, and Fridays from 10 December 2019. They can edit the work times in the Shifts panel, as shown in Figure 5.9. After adding the shifts, the simulation is re-run. The results show that now the project can end ahead of the expected end time, by 8 June 2020 instead of 24 June 2020, showing that adding more shifts can accelerate the project.

So, by going through these what-if scenarios, the Greenhouse project manager can conclude that he can add more shifts to the current calendar, rather than hire more resources, to accelerate the project and meet the deadline.

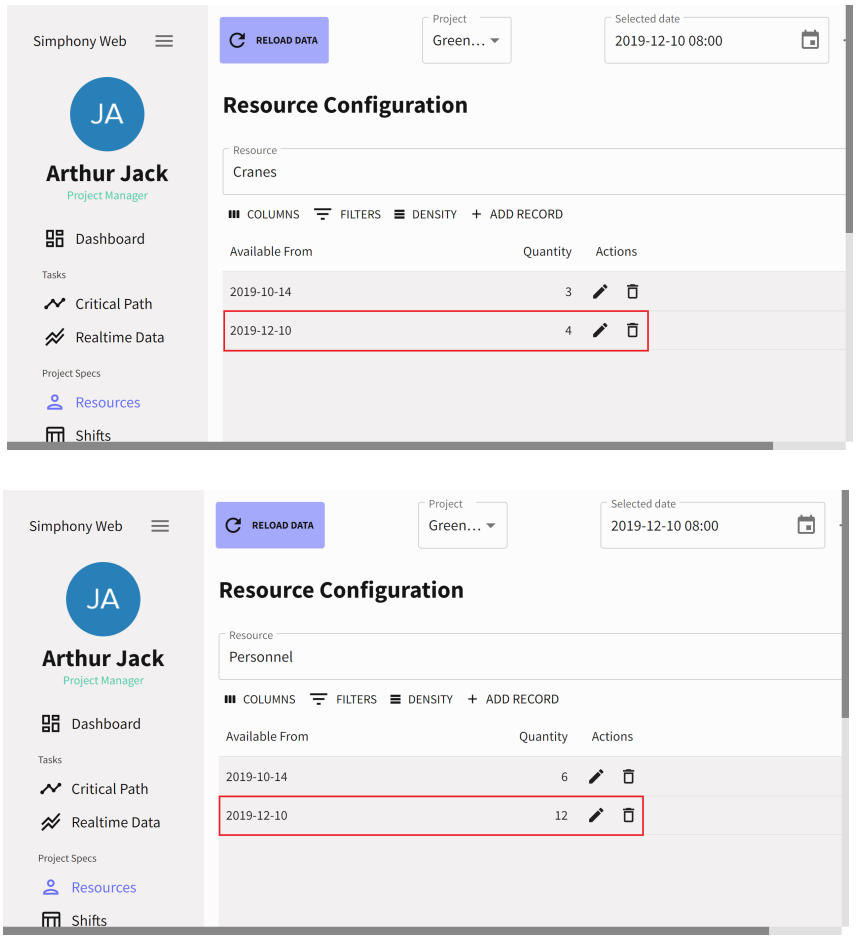


Figure 5.8: Updates done to the resource configuration of the Greenhouse project on 10 December 2019 (Zoomed in for better view).

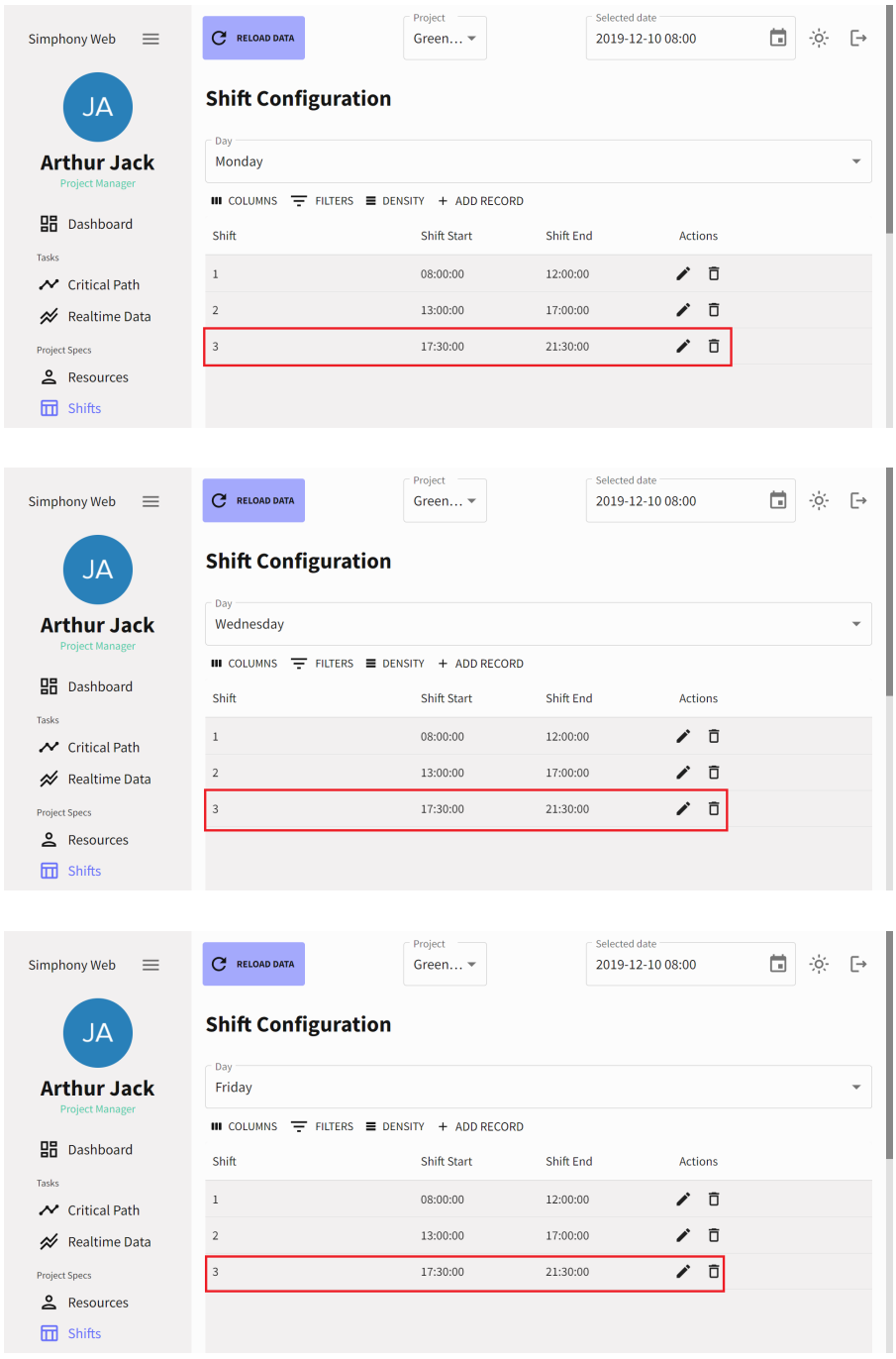


Figure 5.9: Updates done to the shift configuration of the Greenhouse project on 10 December 2019 (Zoomed in for better view).

Chapter 6

Evaluation

6.1 Evaluation of Simulation Results

Q. Are the simulation results consistent between *Simphony Dynamic* and *Simphony*$_{aaS}$?

To evaluate the simulation results generated by our system, we collected a model from the creators of *Simphony Dynamic* . The model represents a real construction project by PCL Construction, with 1691 tasks. It was provided to us as a Microsoft AccessDB file and using a Python script, we transformed it to our model JSON format. We simulated the model on *Simphony*$_{aaS}$ and *Simphony Dynamic* and collected the simulation results for comparison.

We discovered that although both simulations start and end on the same day, the order of task execution was different between *Simphony Dynamic* and *Simphony*$_{aaS}$. We printed out all the calculations happening within the simulations and found out that there were slight differences in floating point calculations, especially during calculations of the remaining man-hours of a task as the simulation progressed. We looked further into the matter and discovered that the floating point representation of .NET was updated in .NET Core 3.0 [31], which was causing this. This change caused the calculations in the simulation in *Simphony*$_{aaS}$ to shift progressively and created a chain reaction that changed the execution order of the tasks. However, the simulated project end dates were unaffected and were the same in both systems.

We concluded that the results are different due to the differences in floating point representation. A possible fix for this issue can be representing all floating

point values in the *Simphony Dynamic* engine with the C# type Decimal rather than Double, which allows better precision for floating point calculations.

6.2 Performance Evaluation

To test how our system performs for models of various complexity and the overhead of $Simphony_{aaS}$ compared to *Simphony Dynamic* , we conducted performance evaluations. All evaluation was done on an AMD Ryzen 7 7840HS CPU with 32 GB of RAM.

6.2.1 Scalability

Q: How does the system perform under different project sizes? What project attributes impact run-time cost?

For scalability evaluation, we reused the Greenhouse project and created two different scenarios: increasing the number of tasks i.e. Greenhouses to be built, but not increasing the number of resources, and the other with resource scaling i.e. increasing the number of resources as needed as we add more Greenhouses. These two scenarios present two cases - a large number of tasks with not enough resources to tackle them, and another with enough resources to handle them. We ran the simulation for a total of 100 iterations, and for each iteration, we increased the number of tasks by 22 and the number of resources by 41 in one scenario (marked with orange in Figure 6.1). In the other scenario, we only increased the number of tasks by 22 while keeping the number of resources to 41 always (marked with blue in Figure 6.1).

Figure 6.1 shows the results of the performance evaluations. We can see that if we do not scale the resources, the runtime increases rapidly and reaches 185.95 seconds for a simulation with 2200 tasks. But scaling the resources keeps the same simulation runtime within 2.66 seconds. This shows that the simulation engine struggles if there is a lack of resources and awaits the release of resources when simulating tasks, hence taking more time. The results also show that the runtime is strongly positively correlated to the number of tasks in the simulation model, with

a Spearman's ρ value of 0.99 for both tests, indicating that the API runtime will increase if the number of tasks increases, regardless of resource scaling. However, for an increase of tasks with no resource scaling, the increase is exponential, whereas for an increase with resource scaling, the increase is linear. Figure 6.1 presents the results of this experiment.

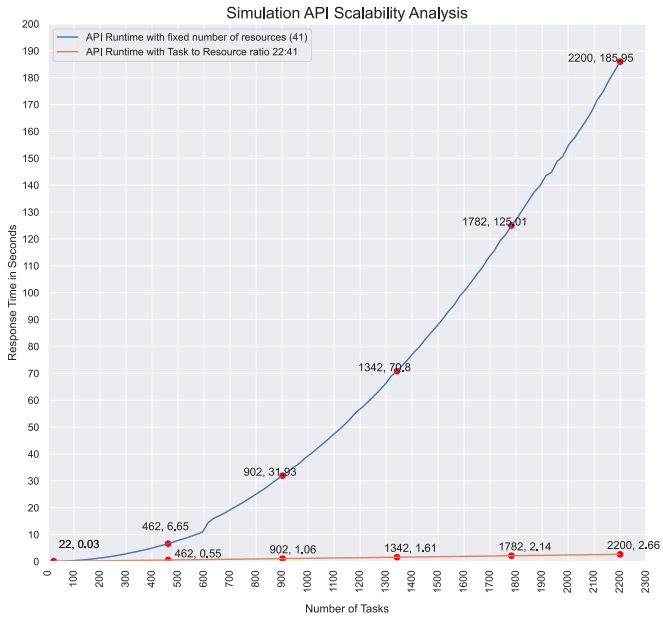


Figure 6.1: Scalability performance evaluation results for the Simulation API

6.2.2 Overhead

Q: What is the overhead incurred by the service-oriented infrastructure on the original simulation cost?

To test the overhead incurred by *Simphony*$_{aaS}$ from the migration process, we used a simulation model with 1691 activities and 180 products, which contains the planning data of a real project by PCL Construction.

For a better testing scenario, we created 18 different chunks of the model, starting from 10 products and their tasks, and iteratively increasing the size to all 180 products and their tasks. Then we took the 18 different chunks and simulated each chunked model 10 times on both our system and *Simphony Dynamic* .

We measured the runtimes at 3 different points (1) The time taken to simulate the model by *Simphony Dynamic* (2), the roundtrip time of the simulation POST API in $Simphony_{aaS}$ (3) the time it takes the simulation microservice of $Simphony_{aaS}$ to simulate the model. The simulation entry point of the *Simphony Dynamic* simulation engine is a C# function named 'Model.Simulate'. We checked the execution time of this function to measure the simulation times in both $Simphony_{aaS}$ and *Simphony Dynamic* . To test the REST API roundtrip times, we calculated the difference between the timestamps of when we sent the request and when we received the response.

Figure 6.2 shows the comparison between the mean runtimes of the 3 scenarios, Figure 6.3 shows the median runtimes of the 3 scenarios, and Figure 6.4 shows the mode runtimes the 3 scenarios.

By observing the results, we saw that for the larger chunks of the model, $Simphony_{aaS}$ performed better, and for the smaller chunks of the model, *Simphony Dynamic* performed better. From here we can conclude that the overhead is much less than the simulation but when the simulation is small they are comparable, so the overhead penalizes the new architecture for smaller models.

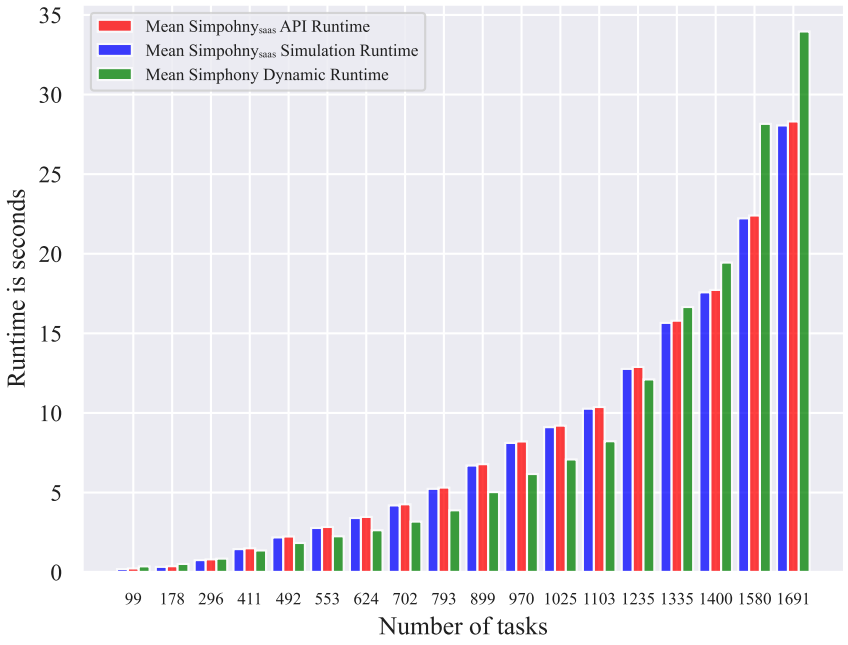


Figure 6.2: Comparative mean runtimes of *Simphony Dynamic* and *Simphony$_{aaS}$*

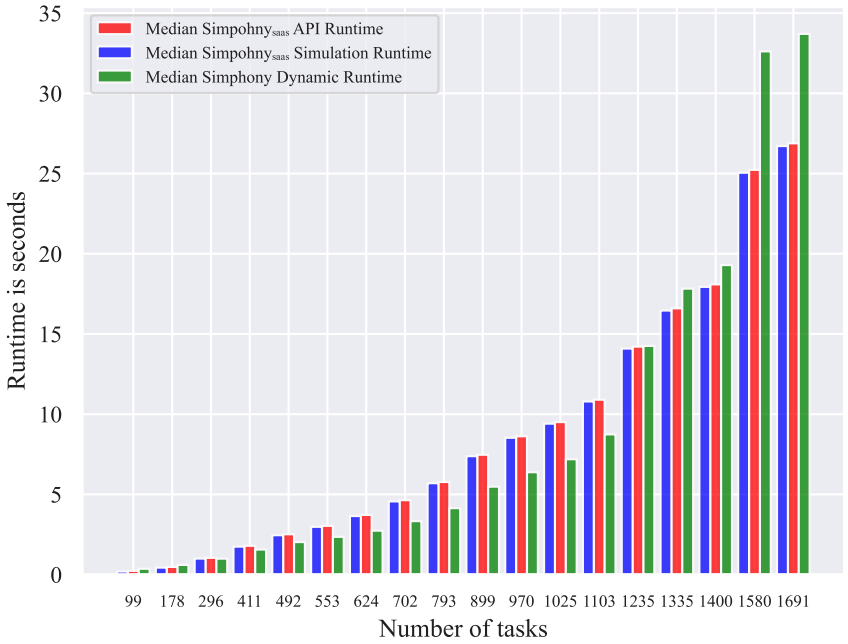


Figure 6.3: Comparative median runtimes of *Simphony Dynamic* and *Simphony$_{aaS}$*

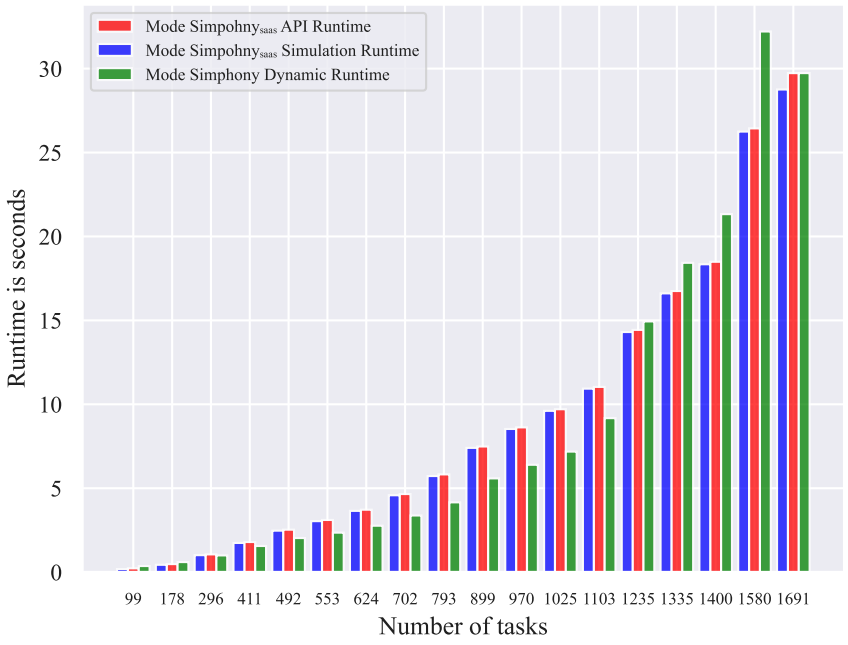


Figure 6.4: Comparative mode runtimes of *Simphony Dynamic* and *Simphony$_{aaS}$*

Chapter 7

Conclusion

7.1 Contributions of *Simphony*$_{aaS}$

In this thesis, we migrated the Simphony Dynamic legacy software (based on .NET Framework 4.6) to a state-of-the-art microservices-based software-as-a-service architecture (based on .NET 8.0). This migration was forced by the fact that Microsoft has announced that they will discontinue support for .NET Framework 4.6. Even if, however, this was not the case, the migration to a web-accessible cloud-based architecture is motivated by two key objectives. First, cloud-based deployment enables easy adoption of the software by users who may be unable to install the legacy desktop application on their own infrastructure. Second, cloud-based deployment enables performance scalability by virtue of the fact that the software runs on state-of-the-art infrastructure and can potentially access increased computational and storage resources.

The migration process enabled several key functional improvements.

- We aligned the project specification model to Digital Construction Ontology, which increased its potential for interoperability and extensibility. In parallel, we migrated the system database to GraphDB, a database that supports the management of RDF triples (in terms of which DiCON is described) and the efficient implementation of a variety of queries of interest.

- We migrated the original Simphony Dynamic user interface to a web-based Model Editing Interface with a similar look and feel.

- In addition, we developed two new user interfaces, conceived to meet the needs of users from different levels of the construction process and present reports and data based on their role in the construction process. The project management interface assists the project manager in exploring what-if scenarios like modifying resources or shifts for better decision support. A new mobile application, and a new REST API, were developed to facilitate real-time data collection by crew leads on the site.

- Lastly, our most important contribution is adding the feature of model re-simulation from any point by automatically adjusting the model parameters. The process is made seamless as the real data is persisted in our system and the user has the option to select this point in time from the project management dashboard. This makes the simulation model reusable throughout the whole project without any manual intervention.

Together, these contributions address some of the problems that were hindering *Simphony Dynamic* from being more widely adopted by the industry.

7.2 Future Work

Future work can focus on integrating the system with popular project management software used in the construction industry e.g. Microsoft Project, and Primavera P6. Most construction companies use these software to plan and schedule their work, and provisions to directly import data from them into our system will greatly increase the chance of industry adaptation.

More what-if scenario exploration can be added to the Simphony Project management dashboard based. The current system does not calculate cost estimation but it can be implemented to make the simulation results even more useful.

Additional Construction factors like weather, risk, material, and equipment can be incorporated into the simulation engine to represent construction sites accurately.

Another possibility is creating machine learning models to automatically learn the on-site progress and report it to our real-time data tracking API. This will automate the data entry and can supplement the mobile application.

References

[1] S. K. Sears, R. H. Clough, and G. A. Sears, *Construction project management: A practical guide to field construction management*. John Wiley & Sons, 2008.

[2] P. D. Galloway, "Survey of the construction industry relative to the use of cpm scheduling for construction projects," *Journal of construction engineering and management*, vol. 132, no. 7, pp. 697–711, 2006.

[3] Microsoft, *Microsoft project*, 2024. [Online]. Available: `https://www.microsoft.com/en-us/microsoft-365/project/project-management-software`.

[4] Oracle, *Primevera p6*, 2024. [Online]. Available: `https://www.oracle.com/ca-en/construction-engineering/primavera-p6/`.

[5] D. W. Halpin, "Cyclone–method for modeling job site processes," *Journal of the construction division*, vol. 103, no. 3, pp. 489–499, 1977.

[6] J. C. Martinez and P. G. Ioannou, "General purpose simulation with stroboscope," in *Proceedings of Winter Simulation Conference*, IEEE, 1994, pp. 1159–1166.

[7] D. Hajjar and S. AbouRizk, "Simphony: An environment for building special purpose construction simulation tools," in *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 2*, 1999, pp. 998–1006.

[8] S. AbouRizk, "Role of simulation in construction engineering and management," *Journal of construction engineering and management*, vol. 136, no. 10, pp. 1140–1153, 2010.

[9] R. R. Labban, S. Hague, E. Pourrahimian, and S. AbouRizk, "Dynamic, data-driven simulation in construction using advanced metadata structures and bayesian inference," in *2021 Winter Simulation Conference (WSC)*, IEEE, 2021, pp. 1–12.

[10] Y. Zheng, S. Törmä, and O. Seppänen, "A shared ontology suite for digital construction workflow," *Automation in Construction*, vol. 132, p. 103 930, 2021.

[11] D. W. Halpin, "Microcyclone user's manual," *Division of Construction Engineering and Management, Purdue University, West Lafayette, Indiana*, 1990.

[12] Halpin, Daniel W, *Simulation in construction using cyclone and microcyclonee*, 2024. [Online]. Available: https://engineering.purdue.edu/CEM/people/Personal/Halpin/Sim/index_html.

[13] J. C. Martinez, "Earthmover-simulation tool for earthwork planning," in *1998 Winter Simulation Conference. Proceedings (Cat. No. 98CH36274)*, IEEE, vol. 2, 1998, pp. 1263–1271.

[14] J. C. Martinez, "Ezstrobe-general-purpose simulation system based on activity cycle diagrams," in *Proceeding of the 2001 Winter Simulation Conference (Cat. No. 01CH37304)*, IEEE, vol. 2, 2001, pp. 1556–1564.

[15] W. Lu and T. Olofsson, "Building information modeling and discrete event simulation: Towards an integrated framework," *Automation in construction*, vol. 44, pp. 73–83, 2014.

[16] C. D. Pegden, "Introduction to simio," in *2008 Winter Simulation Conference*, IEEE, 2008, pp. 229–235.

[17] F. Peña-Mora, S. Han, S. Lee, and M. Park, "Strategic-operational construction management: Hybrid system dynamics and discrete event approach," *Journal of construction engineering and management*, vol. 134, no. 9, pp. 701–710, 2008.

[18] K. Feng, S. Chen, and W. Lu, "Machine learning based construction simulation and optimization," in *2018 Winter Simulation Conference (WSC)*, IEEE, 2018, pp. 2025–2036.

[19] S. AbouRizk, S. Hague, R. Ekyalimpa, and S. Newstead, "Simphony: A next generation simulation modelling environment for the construction domain," *Journal of Simulation*, vol. 10, no. 3, pp. 207–215, 2016.

[20] G. Lucko, P. C. Benjamin, and M. G. Madden, "Harnessing the power of simulation in the project management/decision support aspects of the construction industry," in *2008 Winter Simulation Conference*, IEEE, 2008, pp. 2479–2487.

[21] P. Benjamin, M. Graul, and M. Erraguntla, "Toolkit for enabling adaptive modeling and simulation (teams)," in *Proceedings of the Winter Simulation Conference*, IEEE, vol. 1, 2002, pp. 763–771.

[22] M. Lingineni, B. Caraway, P. C. Benjamin, and R. J. Mayer, "A tutorial on prosim: A knowledge-based simulation model design tool," in *Proceedings of the 27th conference on Winter simulation*, 1995, pp. 408–412.

[23] Y. J. Son and R. A. Wysk, "Automatic simulation model generation for simulation-based, real-time shop floor control," *Computers in Industry*, vol. 45, no. 3, pp. 291–308, 2001.

[24] Microsoft, *Download .net framework 4.6*, 2023. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet-framework/net46.

[25] Microsoft, *Download .net 8.0*, 2023. [Online]. Available: https://dotnet.microsoft.com/en-us/download/dotnet/8.0.

[26] BIM4EEB, *Bimms toolkit*, 2024. [Online]. Available: https://bim4eeb.oneteam.it/BIMMS/default.aspx.

[27] Microsoft, *Asp.net core free. cross-platform. open source. a framework for building web apps and services with .net and c.* 2023. [Online]. Available: https://dotnet.microsoft.com/en-us/apps/aspnet.

[28] Ontotext, *Graphdb by ontotext*, 2023. [Online]. Available: https://graphdb.ontotext.com/.

[29] J. Batselier and M. Vanhoucke, "Construction and evaluation framework for a real-life project database," *International Journal of Project Management*, vol. 33, no. 3, pp. 697–710, 2015.

[30] Operations Research Scheduling Research Group, *Online consultation of the real-life project database*, 2024. [Online]. Available: http://www.projectmanagement.ugent.be/research/data/realdata.

[31] Gooding, Tanner, *Floating-point parsing and formatting improvements in .net core 3.0*, 2019. [Online]. Available: https://devblogs.microsoft.com/dotnet/floating-point-parsing-and-formatting-improvements-in-net-core-3-0/.