University of Alberta

Improving Dynamic Project Control in Tunnel Construction

by

Hua Xie

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Construction Engineering and Management

Department of Civil and Environmental Engineering

©Hua Xie
Fall 2011
Edmonton, Alberta

ABSTRACT

Project risks and uncertainties cause unpredictable project performance. This research intended to demonstrate that project control can be improved with the proceeding of a project, by exploiting new, available project information, and reducing uncertainties. The purpose of this study was to investigate effective tools and methods for dynamic project control.

The main research work refers to the creation of a project monitoring system, tunnel construction simulation, adaptive modeling, project forecasting and planning, and cost contingency forecasting. A project monitoring system was developed to capture progress data. It also provides project progress data for updating a tunnel construction simulation model, which is built on the High Level Architecture. A Bayesian updating simulation component was developed to automatically integrate progress inputs to update key simulation model inputs. Thus, the simulation model is able to revise the inputs in light of the actual progress, to reflect the latest performance, and to ensure the validity of model inputs. The simulation results are then used to forecast future performance, and plan future work. Systematic project schedules and plans were provided, identifying the production, labour hours, equipment hours, and time to completion, as well as the resource and constraints of each main task. Additionally, this research introduced and applied Value at Risk to forecast project contingency, by utilizing actual daily cost as inputs. The forecasted cost contingency, together with labour hours and equipment hours from the simulation model, provides the basis for project budget planning and allocation.

The web-based project progress monitoring system can reduce the time and effort needed for project control. The simulation model provides dynamic project planning and helps the project management team to gain insight into ongoing projects. The cost contingency forecast method provides opportunities to update contingencies, and allocate appropriate contingencies at project milestones, which allows companies to improve capital fund utilization. This research lays a solid foundation for future endeavours in how to make use of newly available project information to improve project control and planning as the project progresses.

ACKNOWLEDGEMENT

First of all, I sincerely thank Dr. Simaan M. AbouRizk, my supervisor and mentor. I am extremely grateful for his vision, support, encouragement, and guidance for years during my study. His sound knowledge, creative ideas, and academic rigor have led my way to the successful completion of my PhD program, as well as urged me to surpass myself.

I extend my gratitude to Dr. Yasser Mohamed and Dr. Ming Lu for their advice, recommendations, and insightful discussions with me. I am grateful to all members at the Hole School of Construction Engineering at University of Alberta for their invaluable support. I would especially like to thank the programmers who worked with me and contributed to the development of COSYE and the tunnel federation.

Special thanks to the engineers from the Design and Construction Section of Drainage Services at City of Edmonton, and S.M.A. Consulting Ltd. Their continuous support and assistance helped me in collecting and sharing project data, analyzing the construction process, and gaining insight into their operations, which provided a great opportunity to obtain experience and knowledge in tunnel construction.

I would like to express my sincere gratitude to my Mom and Dad. Their sustained patience, encouragement, and support have continuously cultivated my passion and endeavours in pursuing my dreams, including this degree.

Thank you to all the people who helped me through the doctoral program. The study and experience at University of Alberta has not only taught me knowledge and skills, but also fostered my tenacity, rigor, and enthusiasm about work and life.

I would like to end this acknowledgement with the lyrics from one of my favorite songs, *You Learn*, by Alanis Morissette. I believe these lyrics truly illustrate the study and life of my PhD program, and I would like to share them with the ones whom I love and who love me:

"You live, you learn

You love, you learn

You cry, you learn

You lose, you learn

You bleed, you learn

You scream, you learn…

You grieve, you learn

You choke, you learn

You laugh, you learn

You choose, you learn

You pray, you learn

You ask, you learn

You live, you learn…”

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Introduction

Although the purpose of project control is to ensure that a project remains on track, on time, and within budget (Project Management Institute 2004), in reality, the practice is far from perfect, in the construction domain. A US Government Accountability Office study reported that half of the federal highway and bridge projects that it examined had cost overruns of more than 25% (Government Accountability Office 2003). An average of 28% cost overrun was observed by Flyvbjerg et al. (2002), based on the statistical study of 258 transportation infrastructure projects worth US$90 billion in total. Edwards (2009) summarized many federal projects with cost overruns, such as the Virginia Springfield interchange which cost US$676 million, but was estimated at a cost of US$241 million in 1994 (Shear 2002), and the Boston Big Dig highway project, which cost US$14.6 billion, and was estimated at a cost of US$2.6 billion in 1985 (Lewis and Murphy 2003). The Ministry of Statistics and Programme Implementation in India also showed that the percentage of projects with schedule delay fluctuated between 62% and 31.72% yearly from 1991 to 2010 (Narayan 2011).

Lack of timely and effective communication, lack of integration, uncertainty, a changing environment, and increasing project complexity all lead to project change (Naoum 1994). Various project participants work together and generate diverse information, which makes the data collection and transfer process complex. For example, a tunnel construction site involves numerous participants including labourers for construction work, suppliers who provide materials such as concrete liners, technicians who check and/or repair the tunnel boring machine (TBM), surveyors who make sure the tunnel alignment is correct, and so on. Different project participants have different concerns about the project and generate diverse types of data. All kinds of information must be collected and considered as part of the decision making process in project control.

Confliction will be reduced if problems can be identified as early as possible and proactive actions can be made (Kartam 1996). Project management teams should be able to react to changes effectively for the success of the project (Ibbs et al. 2001). The project management team needs to obtain site information in a timely manner to identify problems and create fast resolutions for project control. Information delivery and processing in the construction field is traditionally paper-based; even foremen collect daily progress reports in a written log. It normally takes days to transfer these reports to project managers and other project stakeholders. Effective project control requires the project management team to take corrective actions when variance from the plan is identified in actual performance.

Advances in computer techniques bring new opportunities for improving project management and control. Currently, companies rely heavily on computer software in project management and control. According to a Construction Financial Management Association (CFMA) survey of a selected list of contractors of CFMA members (approximately 4,000), from 1998 to 2002, 80% of the respondents utilized project management and control software in their daily business management, on average. 59% of the market share of management software is held by MS Project, SureTrak, and Primavera Project Planner. However, many companies prefer to develop their own customized control systems to match their own project properties, management requirements, and company culture.

This research integrates High Level Architecture (HLA)-based discrete event simulation (DES), automated data collection, automated simulation model input updating, Bayesian updating technique, the theory of Last Planner System, and Value at Risk, to achieve dynamic and collaborative project control in a distributed tunnel simulation model. It also utilizes the tunnel simulation model in integrating data collection of project progress data, analyzing progress data and their impact on future performance, and producing updated project cost and schedule plans based on actual progress data. It aims at improving schedule and cost controls for a project management team, providing insight into ongoing projects and helping the project management team make faster and better decisions.

1.2 Background

The project monitoring and control process includes all activities to track, review, and regulate the progress and performance of the project, identify necessary changes to the project plan, and initiate corrective actions, according to the Project Management Institute (2008). Hinze (1998) proposed that project control consists of two basic components: monitoring and control, where monitoring means collecting information from a project and discovering what is actually occurring, and control includes responding to the collected information. Mantel et al. (2010) also defined monitoring as "the collection, recording, and reporting of project information that is of importance to the project manager and other relevant stakeholders," and stated that control refers to using "the monitored data and information to bring actual performance into agreement with the plan."

Project monitoring and control interact with other processes throughout the course of a project, as shown in Figure 1-1 (Project Management Institute 2008). The project monitoring and control group, along with the initiating process group, planning process group, executing process group, and closing process group, constitutes the overall project management process (groups) (Project Management Institute 2008). The planning process provides the project monitoring and control process with project baseline plans and project documents; as the project progresses, the project baseline plans and project documents will be updated. Project management involves continually planning the work, measuring the progress, identifying gaps between the actual progress and the plan, taking corrective action to bring the project back on track if required, and replanning when needed (Mantel et al. 2010). Mantel et al. (2010) proposed the "plan-monitor-control cycle," which carries on until project completion.

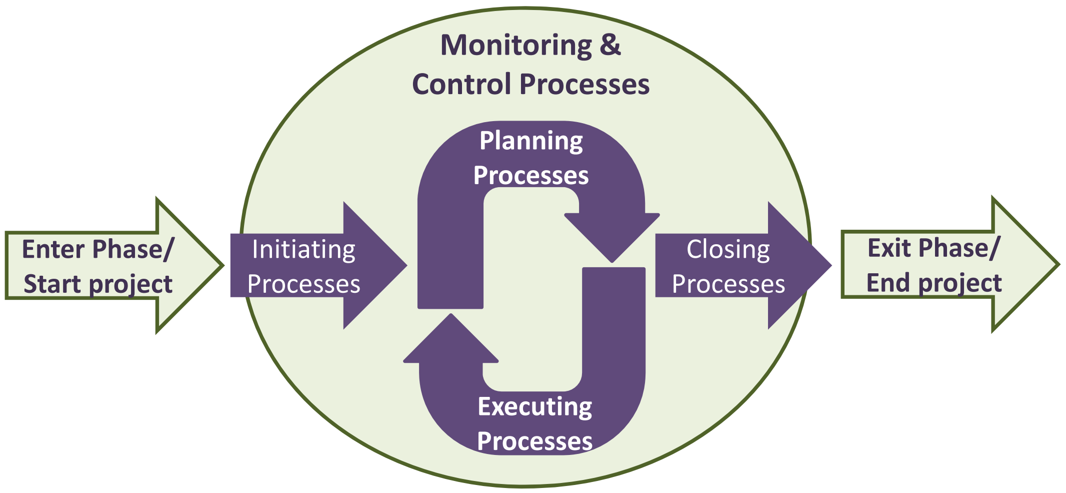


Figure 1-1 Project management process groups (Revised from Project Management Institute 2008)

Although there are different definitions of project control, in this research it refers to the interactive process of planning, monitoring, and regulating projects during project execution. This research will apply different tools and methods to improve project control directly or indirectly, in data collection, project monitoring, assisting decision-making for corrective actions, and in dynamic planning.

The earned value method (EVM) and the Last Planner® System (LPS) are recognized as two prevalent theory systems of project control by Kim and Ballard (2010), who presented a comparison of these two methods. EVM has been widely applied as an integrated project control tool (Fleming and Koppelmam 2000). EVM integrates project scope, cost, and schedule measures to measure project performance and progress, and can be applied to all projects in any industry (Project Management Institute 2008). More details about EVM were presented by Fleming and Kofflema (2010), and Marshall (2007). A recent example of applying EVM in project control was presented by Hanna (2011), but it does not support reasoning to explain either unacceptable performance or suggest corrective action. Nassar (2005) also criticized EVM-based forecasting for assuming that future performance will be a linear continuation of the past, which means that project uncertainty and changes are not dynamically incorporated.

LPS serves as a production planning and control tool by improving work flow reliability (Ballard 2000). A recent example of applying LPS in project control was presented by Nieto-Morote and Ruz-Vila (2011). One of the key issues for implementing LPS is to produce reliable work plans (Ballard 1994; 2000; Ballard and Howell 1998; 2003). Mitropoulos and Nichita (2010) also concluded that reliable work assignment was critical in LPS-based production control, according to lessons learned from a residential development project. LPS focuses on schedule and production planning, but does not support project cost control.

Considering the benefits and defects of EVM and LPS, project control can be improved through integrated project cost and schedule control, reliable performance forecasting and future planning, and assisting decision-making in taking corrective action. Some companies use both LPS and EVM for project control. Kim and Ballard (2010) analyzed the implementation of LPS in 22 projects, and found that 7 out of 22 projects used the EVM along with LPS.

Database applications have been widely developed and used as project control systems. Some of these platforms have been used in conjunction with knowledge-based expert systems (Diekmann and Al-Tabtabai 1992), while some researchers (Chan and Leung 2004; Chou and Chong 2008; Li et al. 2006; Moselhi et al. 2004) have applied web-based systems for convenient data collection and data sharing over the Internet. Automated data collection (Ghanem and Abdelrazig 2006; Navon 2005) is another trend for improving project control. Database applications are capable of dealing with large amounts of data, and are powerful tools for data analysis (as seen in these examples). However, these systems do not account for uncertainties, project logic, and resource constraints, all of which can help give insight into the project and improve the project control system.

Advances in computer techniques, such as distributed simulation and modeling, provide new opportunities for dynamic and collaborative project control. Computer simulation is able to incorporate construction project logic, work sequence, resource constraints, and to produce project cost and schedule. Computer simulation techniques can help project practitioners gain insight into the underlying project, and possibly contribute to reduced project costs, improved project quality, and shortened duration (Abourizk 2010). A successful example of applying simulation for planning tunnel construction projects was presented by Al-Bataineh (2008). Database applications can also provide powerful data analysis function and generate various analytical reports.

1.3 Tunnel construction process

A tunnel project starts with the selection of the site location. During this stage, a project team takes into account a number of factors, including availability of water supplies, electricity, easy access points, and space to store material on site. Then, a construction crew will excavate a working shaft, a tail tunnel, and the undercut, in sequence. After these are completed, the construction crew can start excavating the tunnel.

A tunnel can be excavated by TBM, or by a crew using tools like jackhammers, i.e. hand tunneling. While excavating a tunnel, a train consisting of a few muck carts is employed to remove dirt from the tunnel face to the bottom of the working shaft. Then, a crane is used to lift muck carts one at a time and dump dirt above ground. Sometimes, the construction crew will install a switch to accommodate two trains in a tunnel, so as to accelerate the tunneling operation; this process will be referred to here as "full-capacity tunnelling." One train travels to the TBM and unloads the material car, which usually represents concrete liners or rib and lagging. Then, the TBM starts to excavate. When the train is full, it starts traveling toward the working shaft at the same time as the TBM starts installing the material to support the tunnel. Once the full train reaches the working shaft, the other train (if empty) starts travelling towards the TBM. When that train reaches the TBM, it begins excavating the next section, if the TBM has completed material installation; if not, the train waits. When the train reaches the working shaft, the crew checks whether the previous train has finished unloading. If it has, the train can start unloading the dirt and loading material; if not, the second train should wait until the previous train finishes.

This operation repeats until the TBM reaches the end of the tunnel. During that time, a removal shaft is constructed in the same way as the working shaft. A removal shaft is used to remove TBM from tunnel at the end of a project. The construction process described above is modeled in the tunnel federation.

1.4 Research scope

This research tries to improve project control for tunnel construction, in both cost and schedule, through computer simulation, tunnel progress monitoring and analysis, automated model inputs updating, etc.. Computer simulation is a fundamental tool in this research. This research applies HLA in developing the tunnel simulation model, and investigates the integration of different simulation modules, varying from data collection, data analysis, discrete event simulation, and reporting.

The timely collection of and integration of progress data into a simulation model provides the source data for continuously updating simulation model inputs as a project proceeds. Short-term and long-term work plans are automatically produced from a reporting module in the simulation model. The plans show the expected activities and their progress, resource constraints, as well as labour hours and equipment hours. Thus, both schedule and cost estimates are produced from simulation.

The project cost produced from the simulation model is measured in labour hours and equipment hours, and is considered the base project cost. Project cost contingency is the reserved money to cover unforeseen risks, in addition to base project cost. This research also investigated new methods for forecasting project cost contingency in dealing with uncertainties and risks through project execution. The forecasted contingency, along with the forecasted direct cost from the simulation model, provides the basis for planning and allocating the project budget.

This research demonstrates how to improve dynamic project control through project execution. It developed a HLA-based discrete event simulation model to simulate tunnel construction process; developed a web-based database application to facilitate data collection; applied statistical method to update the DES model input, and continuously produce tunnel construction plans; and applied a method to forecast project cost and contingency. This research enables the tunnel simulation model to dynamically evolve with components developed for different participants, yet is combined in a structured manner. A new quantitative method for updating cost contingency was presented and demonstrated through a case study of tunnel construction.

1.5 Research objective

The primary goal of this research was to facilitate dynamic project control and planning through project execution. The research work covers monitoring, analyzing, and modeling tunnel construction process and planning production, and cost for tunnel construction projects. To achieve this goal, the following sub-objectives were defined:

1) Design and develop a discrete event simulation model of tunnel construction by accounting for resource constraints, material procurement and supply, and project logic, as well as identifying the activities and project logic influencing productivity for tunnel construction;

2) Integrate various simulation components into one simulation model that is expandable, enables collaborative work among model developers, facilitates the automation of integrating progress data into simulation inputs, and improves planning;

3) Explore the use of Bayesian updating technique to enhance the forecast of TBM penetration rate and TBM breakdown;

4) Enhance the quality of cost contingency forecast using VaR method;

5) Automate data acquisition by collecting project progress data from scattered construction sites;

6) Integrate the project progress into a simulation model;

7) Utilize both historical data and as-built project progress data in predicting and planning project production and cost for ongoing projects;

8) Produce and update project milestone plans, look-ahead schedules and weekly work plans which summarize project status, outline ongoing activities, and identify resource requirements for each activity through project execution;

9) And finally, plan and update cost contingency by periods through project execution.

1.6 Research methodology

In order to accomplish the above-mentioned objectives, the following approach was taken:

1) Establish a good understanding of construction process, construction methods, and project logic by consulting project managers and field engineers, reading technical documents, and visiting construction sites at the City of Edmonton Asset Management and Public Works Department.

2) Employ CoSyE, an environment implementing the High Level Architecture for simulation model development, to design and develop a discrete event simulation model of tunnel construction. The following steps were identified and followed:

   a. Analyze and abstract the product (a tunnel) into components, including shafts, tunnel sections, undercut, flow control system, and others;

   b. Identify resource constraints during tunnel construction;

   c. Analyze model inputs and outputs to design model components;

   d. Figure out the function of each simulation component, and specify the information exchange between them;

   e. Develop and realize the function by programming in C# under CoSyE.

3) Develop a web-based database application to capture, analyze, store, and review project progress data, and automate the process of integrating project progress data into a simulation model.

4) Apply Bayesian inference to incorporate as-built project progress data and historical data to update key resource productivity and failure rate, which can then be utilized in a simulation model.

5) Utilize simulation results to produce and maintain project plans and schedules, in accordance with the LPS, through the whole course of project execution.

6) Exploit Value at Risk to integrate actual project cost to forecast and update project cost contingency by periods.

1.7 Thesis organization

This thesis consists of eight chapters, as detailed below.

Chapter 2 summarizes the literature review of existing research related to the research area of this thesis. The research area contains discrete event simulation, HLA-based simulation, simulation model input management, project control systems, construction planning and scheduling, and cost contingency. This chapter also identifies and analyzes challenges in these research areas and leads to the research of this thesis.

Chapter 3, Conceptual Model for an Integrated Tunnel Simulation System, tries to present the overall research of this thesis in a systematic way. It describes the conceptual model of the overall research, the methodology, and the relationship among different modules.

Chapter 4 introduces the implementation of the integrated tunnel simulation system. It discusses the development process, the main components of the simulation system, the improvements of the simulation model compared to previous research, and the application of this simulation system to an actual ongoing project.

Chapter 5 introduces how to apply Bayesian updating to enhance the quality of TBM penetration rate forecast using actual project information. The theory of Bayesian inference and conjugate distributions, and the Bayesian updating federate that implements the Bayesian inference are introduced.

Chapter 6 introduces a web-based project monitoring system and how it provides progress data into a simulation model.

Chapter 7 describes the proposed quantitative method to forecast project cost contingency. The approach, modeling steps, applicability, advantages and disadvantages of the proposed method are discussed in detail, and the model validation is demonstrated in a tunnel construction project using actual cost records.

Chapter 8 concludes the thesis with a summary along with a description of the contributions and recommendations for future improvement of this research.

CHAPTER 2 LITERATURE REVIEW

2.1 Discrete event simulation in construction

Discrete event simulation has been successfully applied to modeling construction operations as an effective tool to assist decision-making in a wide range of operations in construction (Abourizk 2010), since the first construction simulation tool, CYCLONE, introduced by Halpin (1977). Many other simulation tools were later developed to model various construction operations, such as STROBOSCOPE (Martinez and Ioannou 1994), Simphony (AbouRizk and Hajjar 1998), and so on.

These tools allow users to model various construction processes with different resource combinations and resource constraints. Example applications using these systems in tunnel construction involve predicting tunnel advance rate in soft rock with CYCLONE (Touran and Toshiyuki 1987), forecasting soil types and soil families along a tunnel path with Simphony (Ruwanpura Arachchige 2001), and planning tunnel construction by modeling different construction scenarios with Simphony.NET 3.5 (Al-Bataineh 2008). However, the applicability and efficiency of these models are constrained to tackling projects which are well-defined and do not occur in a collaborative environment (AbouRizk et al. 2009). These construction simulation tools only allow the development and run of a simulation model on a standalone computer in a specific environment such as Simphony; model developers must collect and set up all model inputs.

A literature review shows several simulation models in tunnel construction that have been developed using these construction DES tools. As tunnel construction operations are normally linear and repetitive, they are especially suitable for DES. Touran (1987) applied CYCLONE to predict the advance rate of a small diameter tunnel in soft rock. Ruwanpura (2001) developed a special purpose simulation (SPS) template for tunnel construction operations, which proposed and implemented a new analytical model to predict the soil types and soil families along the tunnel path. Integrated simulation methods were then implemented in the SPS template to determine tunnel productivity. Chung, Mohamed, and Abourizk (2006) applied Bayesian updating methods into a tunnel simulation model and then developed and implemented a simulation-based productivity model for utility tunnel construction operations in order to identify the effects of uncertain factors, improve the prediction of tunnel boring machine penetration rates, and predict productivity under various project circumstances and soil conditions. A general purpose simulation was developed using Simphony for modeling space, logistics, and resource dynamics with genetic algorithms for optimizing the layout based on various constraints and rules (Zhou et al. 2009;

Zhou et al. 2008). Marzouk et al. (2010) applied simulation for planning microtunnels projects and estimating project time and cost for construction. In tunnel construction application, these simulation models have mainly been employed to improve prediction of the productivity or advancement rate in real world situations.

Tunnel simulation models also help engineers make better decisions in project planning. Likhitruangsilp and Ioannou (2003) presented a stochastic methodology, based on DES, to evaluate tunneling performance. Al-Bataineh (2008) employed the concept of scenario-based project planning, following an approach recommended by FIATECH for construction project planning, and incorporated various project information including cost, schedule, productivity, weather effects, material supply, and dirt handling. He developed a method for establishing communication between different simulation templates or simulation elements, deploying these within a simulation-based tool for tunnel construction implemented in Simphony.NET. In addition, Al-Bataineh (2008) proposed a multi-user framework for tunnel construction simulation.

However, most of these simulation models have tried to improve project performance before actual site construction commences. Long-term decision-making support from simulation models is seldom discussed in the literature.

As pointed out by Moon and Phatak (2005), simulation models are usually applied before the initial plans or designs of a project are finalized, and these models are not designed for repetitive usage, because gathering and extracting pertinent information for simulation models is time-consuming and laborious (Bengtsson et al. 2009; Boulonne et al. 2010; Moon and Phatak 2005). Still, the data collection process is the most crucial stage through model development (Robertson and Perera 2002). Boulonne et al. (2010) stated that on average, 31% of total project time is used in input data management for projects using DES to analyze and improve material flow in production. Therefore, improved input data collection and analysis can reduce model development cycle time, and can allow simulation models to provide long-term reliable support. The progress monitoring system in this research is developed for this purpose. It tries to automate the process of feeding project progress data into a simulation model.

2.2 HLA-based simulation

Another challenge in construction operation simulation is to feed distributed information to a local simulation model. These well-known construction simulation tools only allow a simulator to develop models on a local computer. This also means that a model developer must collect and set up all model inputs and run the simulation model in a specific environment. When input data is scattered across different sources and locations, data collection become laborious.

HLA is a standard approach for developing integrated simulation systems by constructing simulations composed of different individual simulation components. The Defense Modeling and Simulation Office (DMSO) of the United States Department of Defense (DoD) first developed HLA for defense-related simulation (Defense and Simulation 1996).

HLA was adopted as an open standard (IEEE Standard 1516) in 2000 by the Institute of Electrical and Electronic Engineers (IEEE), and more details are presented by IEEE Computer Society (2010). It is now increasingly being applied in simulation fields such as training, traffic control, construction planning and control, extending its original military applications. It provides an efficient and cost-effective tool for analyzing the targeted system in various ways. HLA is widely used for training, entertainment, virtual shopping malls, and testing or evaluating hardware (Li 2003; Lu 2006; Park 2005).

An HLA-based simulation system is referred to as a federation, and each simulation component is called a federate. A typical federate can be a simulation component, a surrogate for a human user, interfaces to a human user, or a database application that stores information for other federates. In this research, the tunnel construction simulation system is referred to as a tunnel federation.

All federates communicate with each other through the Run Time Infrastructure (RTI) during simulation. The data exchanges between federates are defined in the federation object model (FOM). The FOM specifies the name and attribute(s) of each object class and each interaction class. Federates interact with the RTI according to FOM specification. Each federate can be developed and run at physically distributed places.

The concept of HLA was utilized and implemented in a construction simulation system, the Construction Synthetic Environment (CoSyE) (Abourizk 2006). The CoSyE simulation environment is a .NET implementation of the HLA (High Level Architecture) standard (Kuhl et al. 2000). It provides an environment to develop a variety of DES models for construction operations.

CoSyE is used for developing distributed simulation models and supporting multiple users in a collaborative environment. The CoSyE architecture contains three core components: 1) a Runtime Infrastructure (RTI) Server; 2) an Object Model Template (OMT) editor, the system framework; and 3) modeling federates. The modeling federates can be integral parts developed using CoSyE or external simulation components with specific functionalities. An example application was demonstrated by AbouRizk et al. (2009) and Zhang et al. (2011). CoSyE was used in developing the simulation system in this research.

2.3 Model input management

Effective input management for simulation models has been recognized to be important in repetitive usage of models (Bengtsson et al. 2009; Moon and Phatak 2005; Skoogh et al. 2010). The input data management process covers from the raw data collection from different sources to data process and entering inputs into the DES model (Boulonne et al. 2010). Skoogh, Michaloski, and Bengtsson

(2010) stated that DES is usually applied for special purpose studies, instead of helping daily business for production engineers, despite its power as a tool for efficiency improvements in production.

According to the literature review, model input management and timely updating in the manufacturing industry were discussed in the last few years. Bengtsson et al. (2009) proposed a methodology for Input Data Management (IDM) in discrete event simulation projects which covers the process of identifying and collecting data, and using an IDM software to extract and process the data. Boulonne et al. (2010) proposed a simulation data architecture to facilitate data sharing between data sources and DES models. They developed a data processing tool, a database, and an interface to provide reusable resource event data for sustainable resource information in DES models. Skoogh, Michaloski, and Bengtsson (2010) presented an approach for automated raw data collection and processing for simulation information, to improve the reuse of DES models by reducing the time-consumption for input data management.

Unlike the manufacturing industry, the construction profession usually has to deal with scattered construction sites and labor-intensive projects. Every project has its own properties and uncertainties. Human judgments are always required in determining overall project progress. Automated construction data collection is on the stage of seeking good applications. It is difficult to find a standard way of collecting project progress. Database applications are commonly used as tools for project monitoring and control by construction companies. This study focuses on integrating progress data into a simulation model, and automated data collection is not a part of the study.

2.4 Project control systems

Database applications are widely developed and used as project control systems, since they are capable of dealing with large amounts of data, and are powerful tools for data analysis. Many companies developed their own customized control systems used for assisting project management and control. Web-based database applications provide convenient data collection and data sharing over the Internet (examples in the construction domain can be found in Chan and Leung 2004; Chou and Chong 2008; Li et al. 2006; Moselhi et al. 2004). Other notable examples of project management and control system include automating material procurement (Chan and Sin 2009), tracking and monitoring scheduling information (Jia et al. 2008), measuring work progress through image processing techniques (Memon et al. 2005; Zhang et al. 2009), forecasting project performance (Al-Tabtabai et al. 1997; Hwang and Liu 2005), and automated data collection (Ghanem and Abdelrazig 2006; Navon 2005). However, these systems and tools only focus on specific aspects for improving project control, and lack the ability to present project logic, optimize resource usage, dynamically reflect project changes, and forecast future performance.

Many change management systems have been developed in the construction domain (examples can be found in Charoenngam et al. 2003; Ibbs et al. 2001; Zhao et al. 2010). These studies focus on the systematic management of changes, the document management of change orders, or the prediction of future changes. How the changes will affect an ongoing project in terms of project plans is seldom addressed. Lee et al. (2006) applied system dynamics to analyze the errors and changes of a project, estimated the impact of them in terms of project duration and project schedule, and supported schedule updates during project execution. Rather than analyzing changes, however, this research tries to use adaptive simulation modeling, which make use of project progress data and automatically updates the simulation model to provide dynamic and systematic planning by considering project logic, resource constraints, machine failures, and other uncertainties.

2.5 Construction planning and scheduling

The project planning process interacts with the project control process during project execution (Project Management Institute 2008). The planning process provides project monitoring and the control process with baseline plans, which need to be continuously updated when circumstances change. Many researchers have investigated and demonstrated that overall project performance can be improved through improving planning reliability during the process of project execution (Alarcon and Calderon 2003; Gonzalez et al. 2010; Gonzalez et al. 2008).

A three-level system for construction planning was introduced by Ballard and Howell (1998), consisting of initial planning, look-ahead planning, and commitment planning. The initial planning produces a master schedule of a project, and is considered the highest level of planning among all three levels. The master schedule shows project milestones and is used as the map of project objectives, which other plans should aim to meet eventually. The look-ahead planning details the master plan, which focuses on some specific work items defined in the master plan, and produces a look-ahead schedule. One purpose of the schedule is to initiate a “make ready process,” which leads to commitment planning. Commitment planning produces a near-term work plan.

Look-ahead planning is equivalent to phase scheduling presented by Ballard and Howell (2003), which produces a plan for completing a phase of a product which is ready to be expanded into a lower level plan. The "phase" was defined as a level of the work breakdown structure of a product to be built (Ballard and Howell 2003). The look-ahead schedule will indicate the resources and constraints of activities in the short term. The commitment planning is usually interpreted as a weekly work plan in real project applications. The weekly work plan specifies workable assignments for related responsible parties in a project, defined as physical, specific work by Ballard (1994). The weekly work plan should consider resources and constraints of every assignment to ensure the plan's reliability and executability. This is achieved through a conversation among responsible parties. The weekly work plan also represents a commitment from responsible parties, once it is confirmed by them.

The concept of project planning is adopted by Last Planner® System (LPS), which has been used in many construction companies for improving project control, and consequently improving productivity. LPS also assumes the philosophy that productivity can be improved by better planning, according to Ballard (1994) and Ballard and Howell (2003). Many researchers have reported productivity increase after implementing LPS. For example, Ballard (2000) observed a 10% to 40% increase. More details of look-ahead schedule/plan in LPS are presented in Appendix A.

Previous research (Ballard 1994; 2000; Ballard and Howell 1998; 2003) identified one or more of the following critical issues for assuring the quality of a weekly work plan for LPS: the definition of assignments, the construction logic or work sequence of the assignments, the right amount of assignments, and the soundness or operability of the assignments.

This research attempts to address these challenges in developing short-term plans and schedules and to show that discrete event simulation can serve as a powerful tool for reliable planning. The author identified an interesting relationship between the modeling elements in DES and the planning elements for weekly work plan used in LPS, as shown in Figure 2-1. DES models construction operations as a chronological sequence of events (Pidd 1998). An assignment in LPS can be represented as a well-defined event in DES, and consequently be modeled as a unit in a DES model. The work sequence of the assignments in LPS can be interpreted as the chronological sequence of events in DES, the soundness of the assignments for LPS can be investigated by modeling resources and constraints in DES, and the right amount of assignments is then just the output from the simulation. DES therefore provides a platform for producing reliable plans based on LPS, and the prevailing construction simulation tools, such as Simphony.NET, are capable of such modeling.

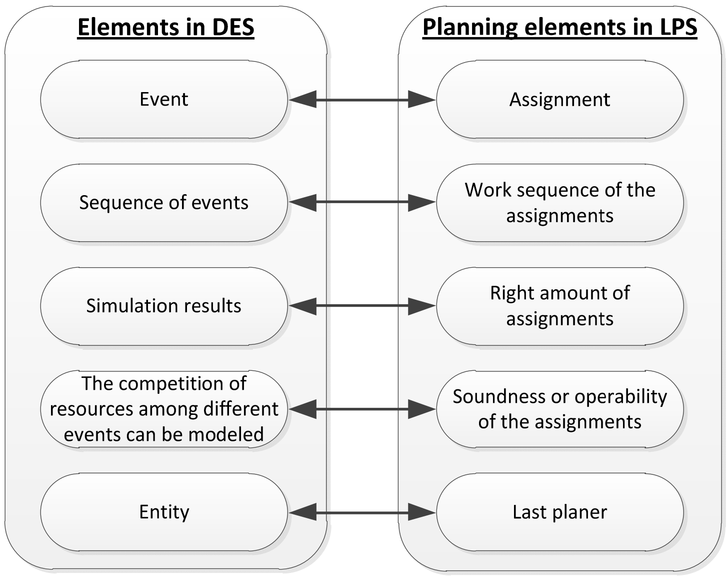

Elements in DES
Planning elements in LPS
Event
Assignment
Sequence of events
Work sequence of the assignments
Simulation results
Right amount of assignments
The competition of resources among different events can be modeled
Soundness or operability of the assignments
Entity
Last planer

Figure 2-1 Elements comparison between DES and LPS

2.6 Cost contingency

“Project contingency” refers to extra funds, budget, or time needed in addition to a project estimate. It is used to cover unforeseen changes during project execution, and shows the stakeholders’ risk tolerances. Contingency is defined as “the amount of funds, budget, or time needed above the estimate to reduce the risk of overruns of project objectives to a level acceptable to the organization" (Project Management Institute 2004). It can be set as a percentage of a project estimate, as a fixed amount, in terms of cost and schedule, or as developed by quantitative risk analysis. Typical tools for risk assessment and contingency estimation include a contingency percentage, risk register, possibility and impact matrix, contingency identified, and three-point and range estimates (Molenaar and Wilson 2009). This research focuses on quantitative cost contingency analysis.

Cost contingency is traditionally estimated as a percentage of the estimated project cost. This method has been criticized as arbitrary (Thompson et al. 1992; Lhee et al. 2009; Cioffi and Khamooshi 2009) and probabilistic models have been introduced to determine an appropriate percentage for cost contingency (Rothwell 2005; Touran 2003). Some research analyzed the contingency of all main activities of a project to better estimate contingency for the overall project (Barraza and Bueno 2007; Günhan and Arditi 2007). However, these models are difficult to validate and in several cases validation was not provided (Rothwell 2005; Touran 2003; Barraza and Bueno 2007; Günhan and Arditi 2007).

More recent studies focused on applying different quantitative methods to determine an appropriate contingency before a project starts (Lhee et al. 2009; Cioffi and Khamooshi 2009; Mak and Picken 2000; Sonmez et al. 2007; Mohamed et al. 2009). They first identified the uncertainty factors (Lhee et al. 2009; Mohamed et al. 2009) or risks (Cioffi and Khamooshi 2009; Mak and Picken 2000; Sonmez et al. 2007) affecting the project, and then developed models taking all factors and/or risks into account to forecast project contingencies. Two studies presented by Molenaar (2005) and Olumide et al. (2010) introduced methods for updating cost contingency through the project design process; both were demonstrated through highway projects. However, all these models rely on original cost estimation, expert judgement, or both for risk analysis. So the validity of these models depends on the accuracy of estimates and unbiased opinions from experts. Tremendous effort is required to collect pertinent and correct information for model inputs, and when the risk environment changes, the process must be restarted to obtain correct information.

Contingencies should be tailored to the risks of the project (Project Management Institute 2004), but updating project contingencies during the project execution process has been only rarely addressed in the literature (Barraza and Bueno 2007). Barraza and Bueno (2007) pointed out the importance of controlling the cost contingency throughout the execution process, but did not refer to how to update that contingency as a project progressed and more information became available to the project. When a project begins to progress and achieves some milestones, the project risks and uncertainties can be identified in more detail. It would therefore be beneficial to update contingencies by periods, rather than deciding on contingencies before a project actually starts and keeping them until project completion.

Cost contingency is determined through cost, and the cost required to accomplish a project includes the cost contingency estimation (Project Management Institute 2004). It is reasonable to expect that the cost contingency of a construction project is put aside as part of the project estimate done before the construction starts. However, if the cost contingency could be appropriately allocated as a project progresses according to a project timeline or a company fiscal timeline, less contingency would be required at the beginning of a project. A large project can last for years, and requires cost contingency measured in millions. Holding the entire project contingency in reserve from the beginning of the project could possibly limit the number and size of projects which a company could invest in and undertake. This puts a constraint on company investment and growth. If a company runs multiple projects simultaneously for years, allocating cost contingencies for each time period, e.g., annually, according to the project tasks and progress, can improve resource utilization and sharing among different projects, and provide a chance to increase investment capacity.

CHAPTER 3 CONCEPTUAL MODEL FOR AN INTEGRATED TUNNEL SIMULATION SYSTEM

3.1 Introduction

Uncertainties and risks are common issues in construction projects which lead to cost overrun and schedule delay. Flyvbjerg et al. (2002) analyzed the cost of 258 transportation infrastructure projects worth US$90 billion, and summarized an average of 28% cost overrun. Virginia Springfield interchange was initiated at a cost of US$241 million in 1994, but ended up at US$676 million by the time it was completed (Shear 2002). Denver International Airport was planned to cost US$1.7 billion in 1989, but resulted in US$4.6 billion in 1995 (Altshuler and Luberoff 2003). The analysis of the Ministry of Statistics and Programme Implementation in India analyzed cost overrun and schedule delay from 1991 to 2010, and identified that yearly cost overruns are between 62% and 12%, and from 62% to 31.72% of all studied projects are behind schedule every year (Narayan 2011). Project control is an essential part of project management which ensures a project remains on track, on time, and within budget (Project Management Institute 2004). Effective project control requires monitoring and evaluating actual project progress, comparing it with the project plan, identifying differences between actual and planned progress, and taking corrective actions if necessary. Collaborative work among various project participants is necessary during these processes, including creating a base plan before a project starts, collecting progress data and analyzing progress, and updating the plan according to project changes and uncertainties.

Construction simulation helps give insight into projects and provides an opportunity to improve project control. It can contribute to reduced project costs, improved project quality, and shortened duration (Abourizk 2010). Nevertheless, special challenges arise when applying simulation to improve project control in the construction profession for the long term. First, construction projects typically involve scattered construction sites and intensive labour work. Subjective judgments are sometimes necessary when determining the overall project progress; it is difficult to collect pertinent and correct information. Second, the lack of tools and methods for timely communication and interaction between model developers and field engineers limits the application of simulation in the project planning phase. Third, the difficulty of acquiring expertise in simulation tools hinders construction practitioners in developing simulation models. Fourth, a simulation model should be adaptively updated to quickly take the uncertainties, as-built performance, and changes into account, because project circumstances change as a project progresses and the uncertainties change correspondingly, which are very important factors when updating a project plan.

This research proposes an integrated simulation-based system for dynamic and collaborative project control, addressing the issues mentioned above. The proposed system provides a convenient way to effectively collect project productivity data and monitor project progress from construction sites; models project logic, work sequence, resource constraints and uncertainty of construction process in a simulation model; integrates latest project performance to the simulation model through project execution; and produces reliable up-to-date project plans and schedules. This leads to long-term support for dynamic project control through project execution. The proposed system is further demonstrated using a tunnel construction project.

3.2 An integrated tunnel simulation system

3.2.1 System overview

This research proposes and develops an integrated system which demonstrates how project control can be enhanced through a dynamic approach to planning that makes use of simulation modeling, Bayesian inference, and automated data collection. It was possible to achieve this goal by utilizing an HLA-based modeling strategy for the project execution. The result is a collaborative approach to project control. The system first collects project as-built progress data through a web-based database application, then incorporates the latest project data with historical data, and analyzes project performance to update model inputs. Consequently, the future performance can be effectively forecasted through a discrete event simulation model. The system continuously collects progress data, updates the simulation model, models uncertainty and resource constraints, and achieves reliable project schedules and plans.

The main conceptual modules and their corresponding relationships are described in Figure 3-1. The project monitoring module is used for collecting and analyzing as-built project information. It is developed as a database application and can reduce the time and effort in using the system for dynamic project control during construction. The information collected will be analyzed by the Bayesian updating module, which is based on the Bayesian updating technique. The Bayesian updating module then incorporates as-built progress data, and adaptively updates the simulation model inputs. The simulation module then uses the updated inputs to simulate the construction process and produce updated and realistic results. The last module produces different project plans based on the simulation results. The process allows the simulation system to take new available information into account and to ensure the validity of model inputs. All system modules can be geographically scattered.

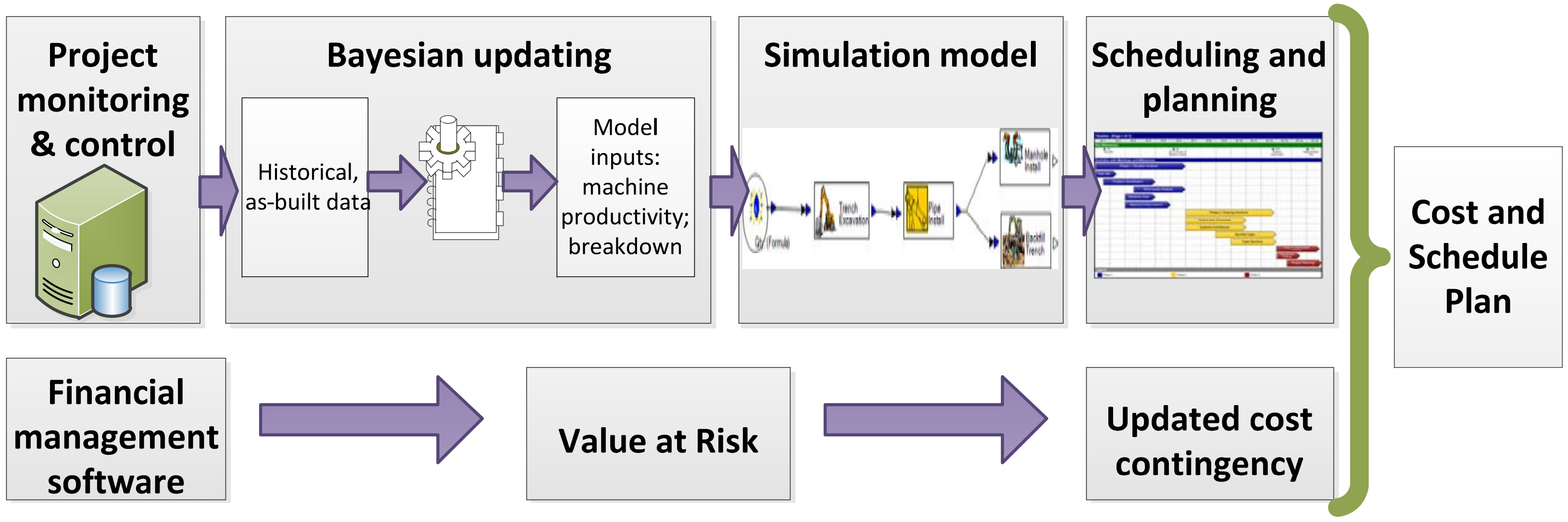


Figure 3-1 A conceptual model for schedule and cost planning

In all, the system attempts to improve project monitoring and control performance by collecting and analyzing project progress, adaptively modeling the construction process, and producing project plans and schedules. The system was developed to assist project management and control using computer techniques and systematic reliable planning. However, a successful implementation of this system depends on different project participants working collaboratively for data collection, simulation analysis, and future planning, etc.

3.2.2 Progress monitoring module

The progress monitoring module was developed as a Client-Server database application for project data collection and progress monitoring. Project progress data is collected through the application and stored using Microsoft SQL Server 2008. This progress data can be reviewed and updated at any time. This module also produces daily reports, weekly reports, and monthly reports for progress review and analysis. Thus, the module can be used as a standalone database application for project progress collection, monitoring, and analysis.

Two types of clients are available, a web-based client and a desktop client. The web-based client allows users to enter project information through personal digital assistants (PDA) such as the BlackBerry® or iPhone®. The web-based client provides users the flexibility to enter data whenever new information is available. This is useful when projects are located in distributed areas and information is gathered from different places. The web client is designed mainly for data entry and review. The desktop client provides a better view of all project information, as well as search functions and filters for data analysis. The desktop solution is designed for project managers who require the ability to review and analyze data, while the web solution is for inspectors who are concerned with capturing data in a convenient way. Details of this database application, such as its reporting function, were presented by Xie et al. (2011).

3.2.3 Bayesian updating module

The Bayesian updating module applies Bayesian techniques to update inputs in the simulation module based on the progress data collected in the progress monitoring module. A model that quickly adapts project parameters to the arrival of new information can increase model validity (Ourdev et al. 2008). The level of overall uncertainty decreases as a project progresses, because the amount of known project information increases and the remaining project tasks decrease. A simulation model which can continuously and promptly incorporate project information is essential in providing long-term support for project control through the project execution phase. As the project continues to change and more information becomes available, model input parameters should be revised and updated according to recent project performance, so as to guarantee the validity of model inputs, and then provide long-term support through project execution (Xie et al. 2011).

3.2.4 Construction simulation module

The construction simulation module is responsible for modeling the construction process, which is TBM tunnel construction in this research. A typical TBM tunnel project consists of constructing shafts and tunnel sections. A working shaft is constructed first, and is used for removing dirt spoil from the tunnel during construction. After finishing the working shaft, an undercut is constructed to connect the working shaft and the main tunnel and to provide enough room for the tunnel crew to work and for TBM installation and dirt removal. The TBM continues to excavate and install segment liners for support. The excavated dirt is transported by a train consisting of a series of muck cars and a material cart. The train travels back and forth between the tunnel face and undercut, where each muck cart is lifted by a crane to the ground level to dump dirt, and the material cart is loaded with segments. The TBM starts excavation again when a train arrives back at the tunnel face. The process repeats until project completion.

3.2.5 Scheduling and Planning module

The scheduling and planning module was developed as a reporting federate in CoSyE, which interacts with the other simulation components as needed to identify the start time and finish time of each activity and then summarize and present information in different schedules and plans. In order to organize and present the data collected from simulation in a meaningful way, the data are first serialized into XML (Extensible Markup Language) format. The XML document is then transformed into browser-renderable HTML (HyperText Markup Language) via XSLT (Extensible Stylesheet Language Transformations) documents, which contain various report formats. The advantage of this method is that the data are separated from the user interface, therefore making their presentation flexible. In addition, XML and XSLT are supported by almost any system with a web browser.

3.2.6 Cost contingency updating

This research tried to forecast project cost contingency, in addition to the direct cost predicted through simulation and presented in the plans and schedules produced in the previous module. This research analyzed planned project cost and actual project cost, and applied Value at Risk (VaR) to establish and update project contingencies. Actual project cost is usually recorded and stored in a financial management system of a company, and used as basis for payroll. Value at Risk provides a probabilistic forecasting of downside risks, and is widely used in financial risk management.

This research makes use of newly available information as the project progresses and provides a chance to allocate appropriate contingencies at different project phases. Different from most other methods that require expert opinions in identifying factors and/or risks, assuming possibility of the factors and/or risks, and estimating their magnitude, this method directly analyzes project cost and earning in dollar values. These cost and earning are considered as the results or reflections of risks through project execution. Compared to other contingency models, this method attempts to minimize arbitrary judgements and proposes a quantitative and uncomplicated model. The method can be widely applied to various construction projects to forecast and update contingencies at project milestones. In addition, the quantitative measure can be simply implemented through a spread sheet, and is simple enough to be understandable and practical for project performance control.

3.3 Summary

Project risks and uncertainties cause unpredictable project performance and project changes. This research hypothesized that as a project proceeds, more information becomes available, uncertainty is decreased, the future performance can be better predicted, and future work plan can be improved. This research tried to capture new available project information and integrate it into a simulation model to produce reliable schedules and plans. The system proposed in this research demonstrated that reliable plans and schedules can be achieved through a simulation model which properly models key machine productivity, its failure rate, the right sequence of work, the right amount of work, and the resource availability of work.

The research tried to demonstrate how project control can be enhanced by using an integrated system which covers data collection, adaptive modeling, discrete event simulation and schedule and plan generation throughout project execution. A web-based project monitoring system was first developed for data collection for projects from scattered locations and information sharing. Bayes' theorem was applied to automatically update inputs for a simulation model. This simulation system then modeled the construction process and the scheduling and planning module continuously produced up-to-date project schedules and plans. The schedules and plans produced can be used as an unbiased and sound basis for planning in LPS, because the as-built project data was continuously utilized and integrated in the simulation model for these schedules and plans.

CHAPTER 4 IMPLEMENTATION OF THE INTEGRATED TUNNEL SIMULATION SYSTEM

4.1 Introduction

The tunnel simulation is developed using the COSYE platform (Abourizk 2006). It is called a tunnel federation, as any simulation system developed based on HLA is referred to as a federation. Each simulation component is referred to as a federate. This chapter discusses all federates that are developed using COSYE in this simulation system. The development of tunnelling simulation system is also described in this Chapter.

This tunnel federation simulates the overall construction process of a utility tunnel. In this study, the process covers the construction of a working shaft, the main utility tunnel, a removal shaft, some access shafts, and an undercut if needed. The main components of the tunnel federation are given in Figure 4-1.

The federation includes model input analysis, construction process simulation, scheduling and visualization, and other support federate groups. Input analysis processes raw data that was collected from the job site and stored in a database. It provides inputs for construction process federates, which simulate the actual construction process. Visualization federates display the simulation process in different formats. The details of the visualization federates were presented by Yang et al. (2010). The support federates try to represent and model aspects of the construction environment that could affect tunnel construction production. For example, the procurement of liners for tunneling is modeled by the procurement federate, and the dispatching and supplying of liners to a specific jobsite is modeled in the supplier federate. The calendar federate models workdays, holidays, shift configuration, and so on. The scenario setup allows a user to configure different construction scenarios, such as number of shafts or tunnel length. The breakdown federate models machine breakdowns.

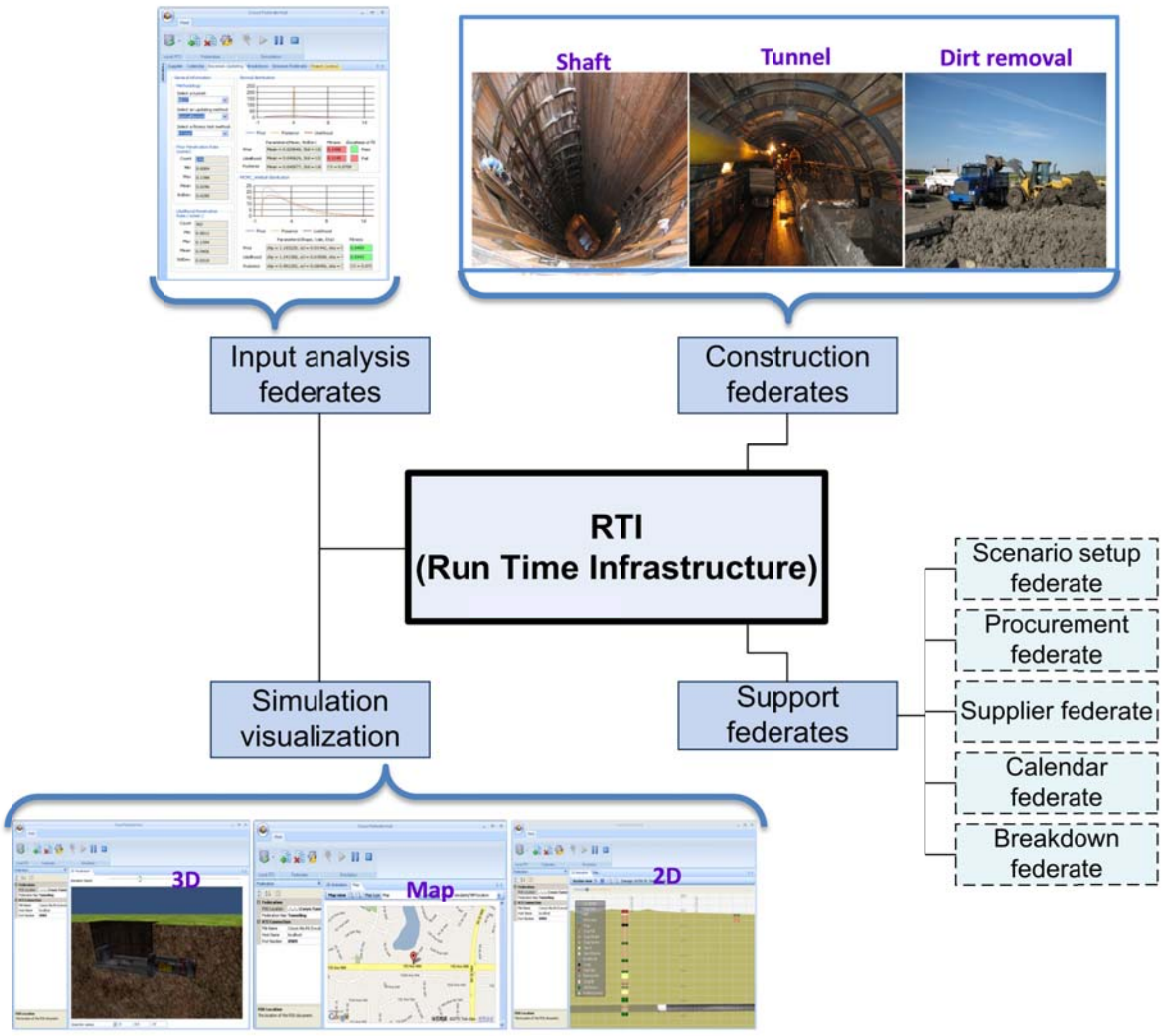


Figure 4-1 Tunnel simulation model

4.2 The development of tunnelling simulation system

The process of tunnelling simulation system development can be divided into four stages, as shown in Figure 4-2. To prepare a simulation model of a tunnel, this research first abstracted the product (a tunnel) into components. The product information relevant to a tunnel includes shafts, tunnel sections, undercut, flow control system, and others. In addition, a thorough understanding of interests from different project participants, various resources required during tunnel construction, and limitations or effects from the surrounding environment all contribute to a reasonable and practical simulation system. Once the preferred observation is achieved, it is time to simultaneously analyze I/O (input/output) and design hierarchical models, which combine tunnelling construction processes, resources, and various related information. The next step is to figure out the function and the information preservation of each simulation module, and the information exchange between them. Good hierarchical simulation models facilitate interoperation among models and improve computational efficiency. After running a simulation model, specific outputs are produced, such as the productivity of processes, the cost of tunnel segments, the utilization of resources, and bottlenecks in the project.

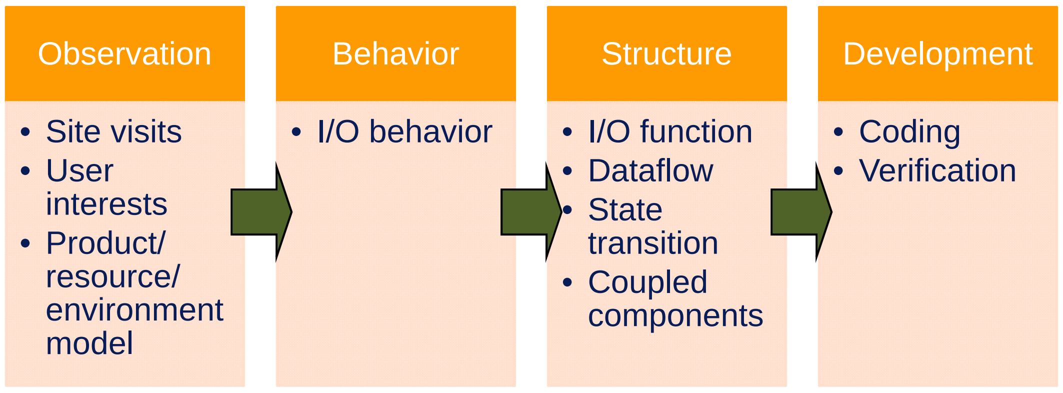


Figure 4-2 Process of tunnelling simulation system development

This research employed hierarchical modeling concepts to facilitate interactions at different levels and by different participants. The principles of hierarchical modeling concepts proposed by Ziegler (1976) were utilized in this research. Ziegler's concepts heavily depend on constructing models using several levels of abstraction. Abstraction in this sense refers to applying a method to a model to reduce its complexity while maintaining its validity in the context of the simulation exercise. As the model is abstracted to a higher level, it is reduced (coupled with other models), thus becoming less complex. The essential steps include partitioning of the models and then coupling (or grouping and simplifying) them in a process of reduction, creating higher-level models (Zeigler et al. 2000).

The proposed simulation system will maintain the lower-level models with appropriate flexible configurations for operational use at the lowest level of the project hierarchy, while abstracting those models at higher levels for managerial purposes. A modeling analysis for tunnel construction is shown in Figure 4-3.

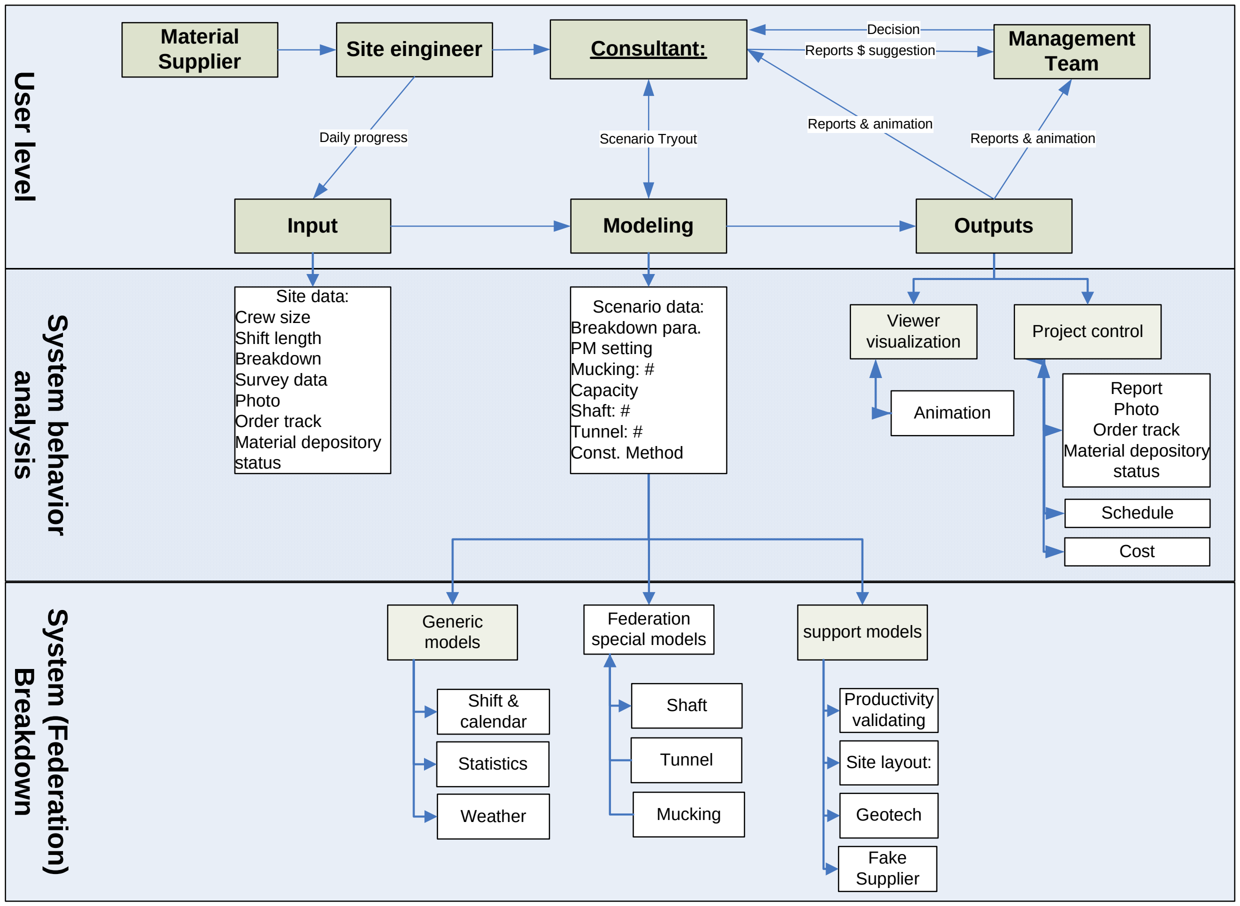


Figure 4-3 Model analysis of tunnel construction

Preliminary research analysis indicated four central pillars in simulation modeling (Figure 4-4); for tunnel construction, these are: tunnel product model, tunnel construction processes, construction environment, and resource management. The following sections explore different components that will be used in this research.

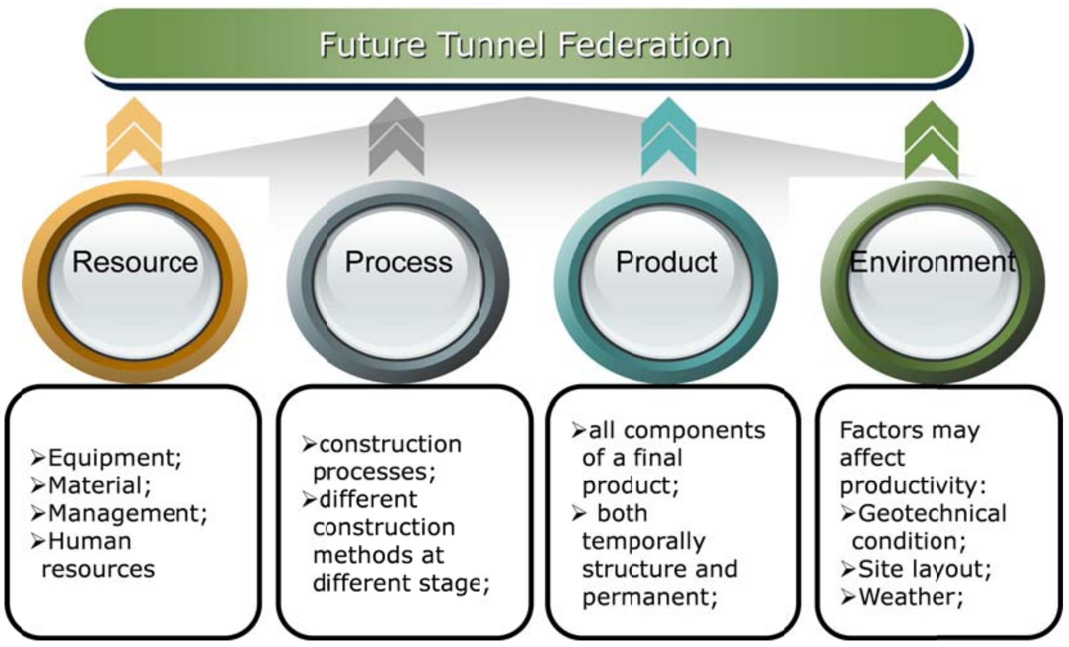


Figure 4-4 Tunnelling federation scheme

A product model is typically a model that expresses a kind of product as a data structure. Tunnel products contain tunnels and working shafts, removal shafts, manholes, and so on. The configuration of tunnels and shafts are also included in the tunel product models. This research attempted to develop a tunnel product model that covers all tunnel components related to tunnel design, construction, and maintenance, and implement it in the integrated simulation. A typical tunnel product model is shown in Figure 4-5.

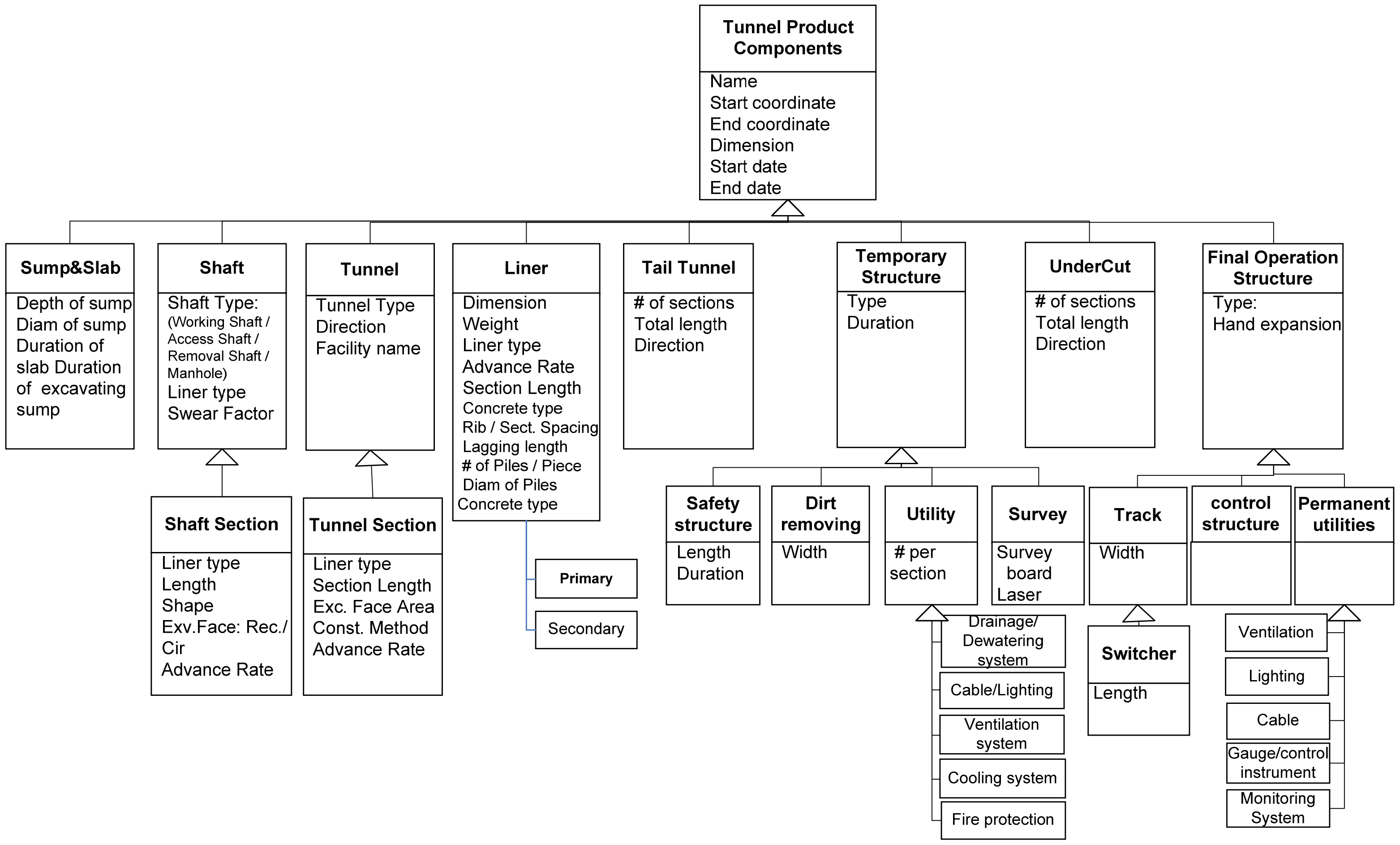


Figure 4-5 A typical tunnel product model

The resources used in tunnelling construction have substantial impact on tunnelling productivity. This research also analyzed the geotechnical or environmental conditions surrounding a tunnel site (construction environment — see Figure 4-6). A resource here refers to any physical or virtual entity of limited availability, or anything used to help tunnelling construction (see Figure 4-7). For example, the TBM used to construct a tunnel, conveyers removing dirt from around the TBM, and the crane lifting dirt from a shaft to the surface are different types of resources for construction.

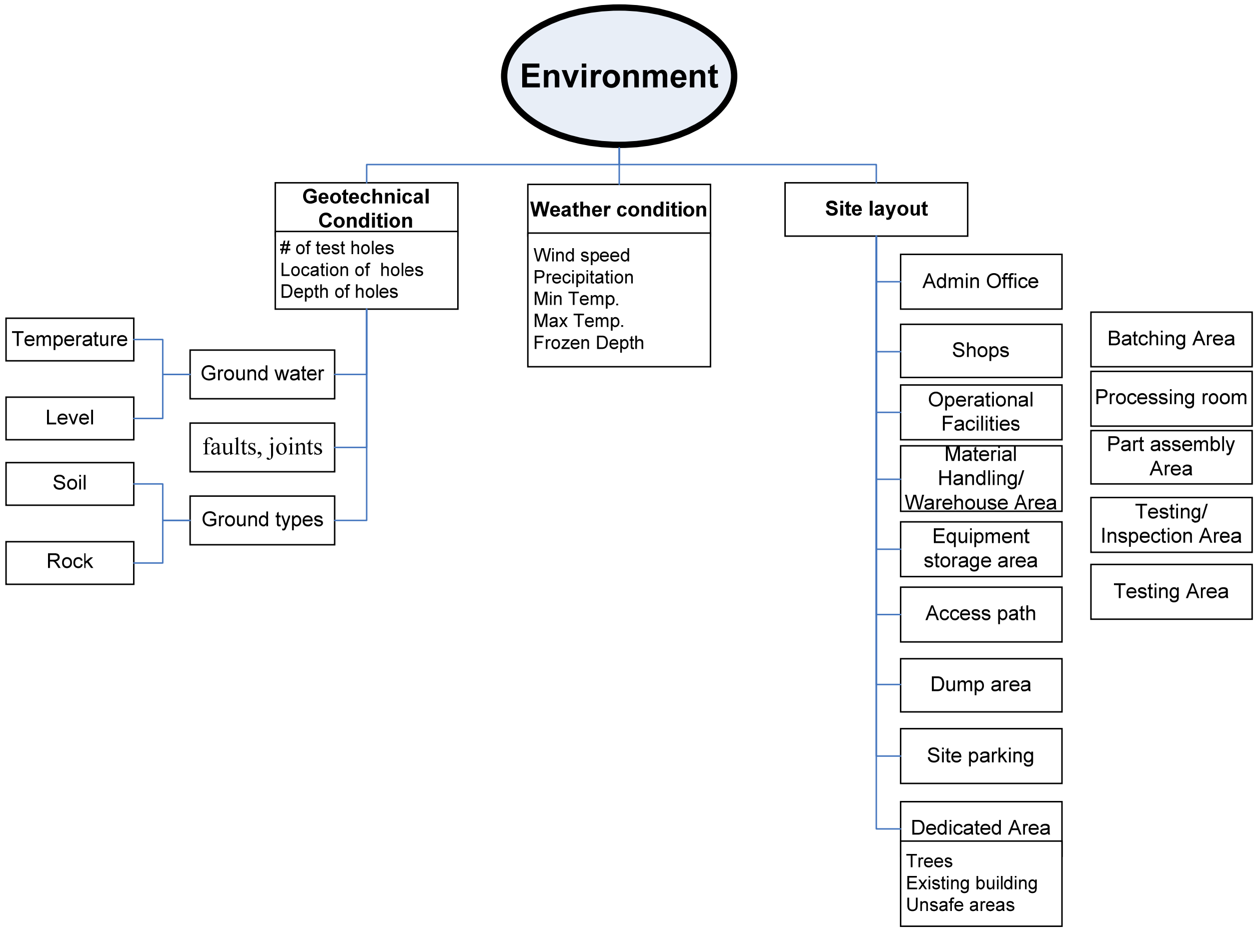


Figure 4-6 Typical factors in tunnel construction environment

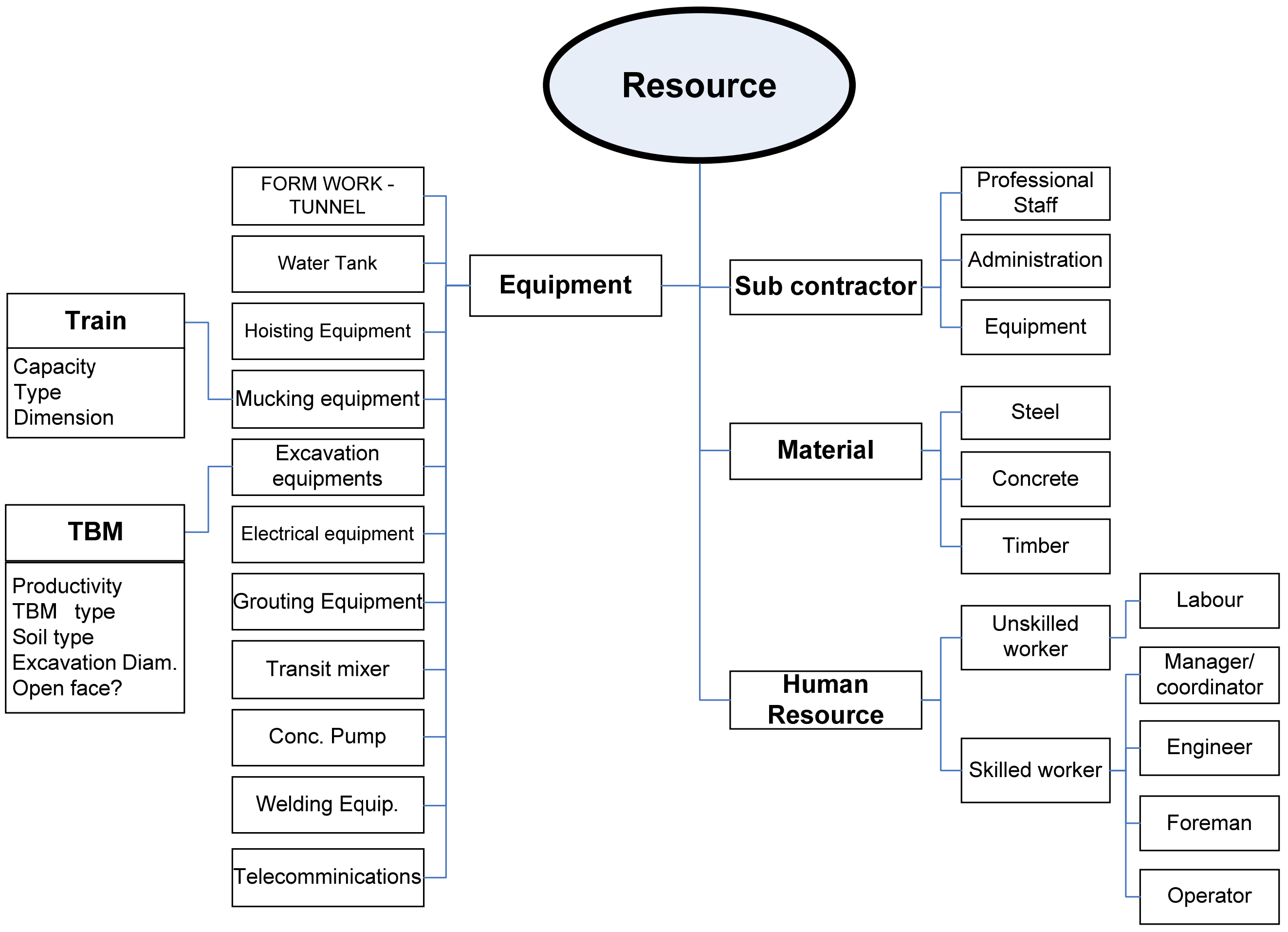


Figure 4-7 Typical resources in tunnelling construction

4.3 Tunnel federation

4.3.1 Overview

The federates in the tunnel federation can also be grouped into three categories according to their functions. The developed tunnel federation and its component federates are described in Figure 4-8. All federates in blue background in Figure 4-8 are directly related to this research.

The first category simulates the tunnel project construction process which includes constructing the tunnel and the shaft, and the related dirt removal process. This tunnel construction is modeled using discrete event simulation technique, and is considered as the core component in the federation. The modeled activities that are related in TBM tunneling are described in Figure 4-9.

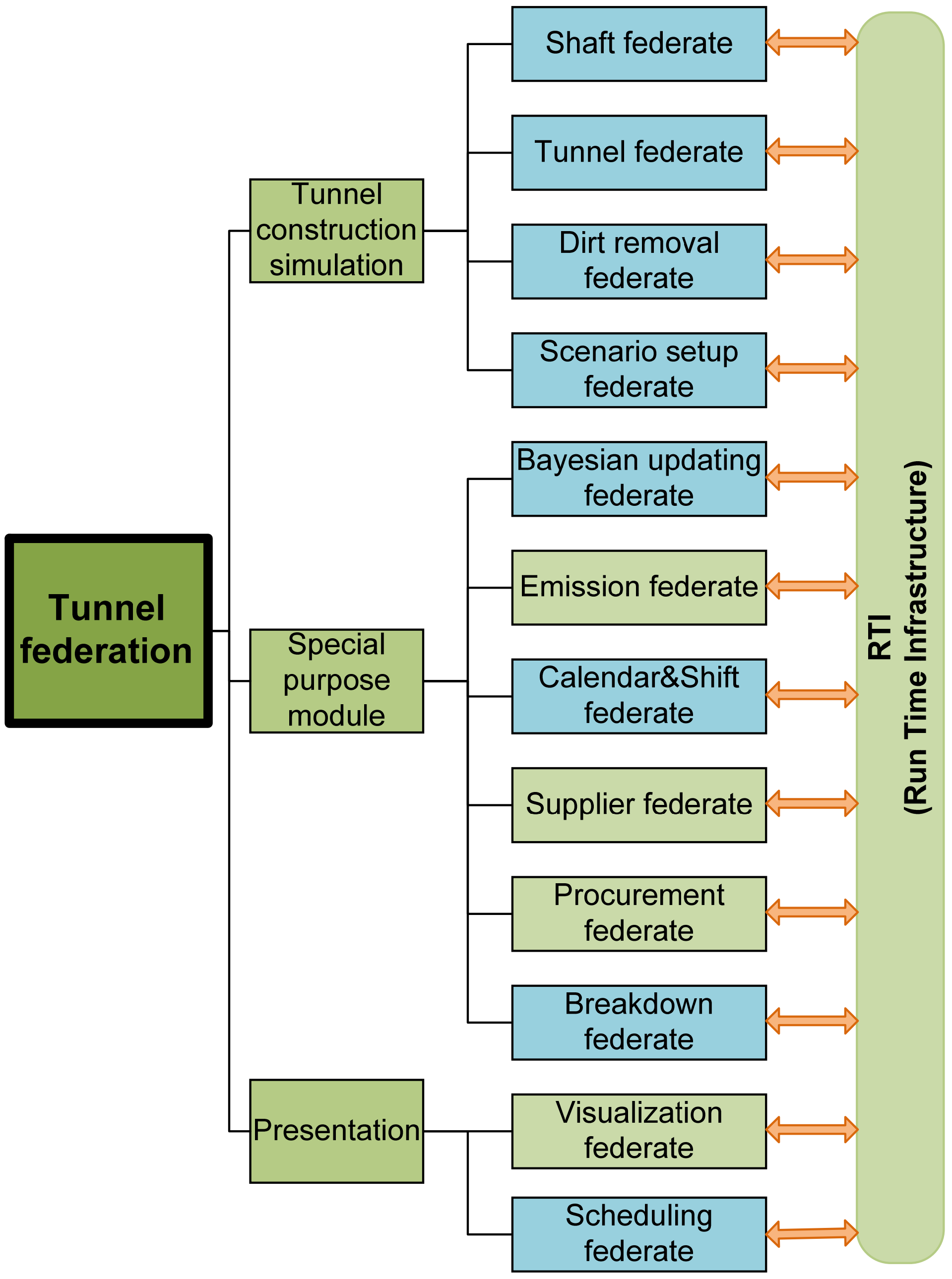


Figure 4-8 Tunnel federation architecture

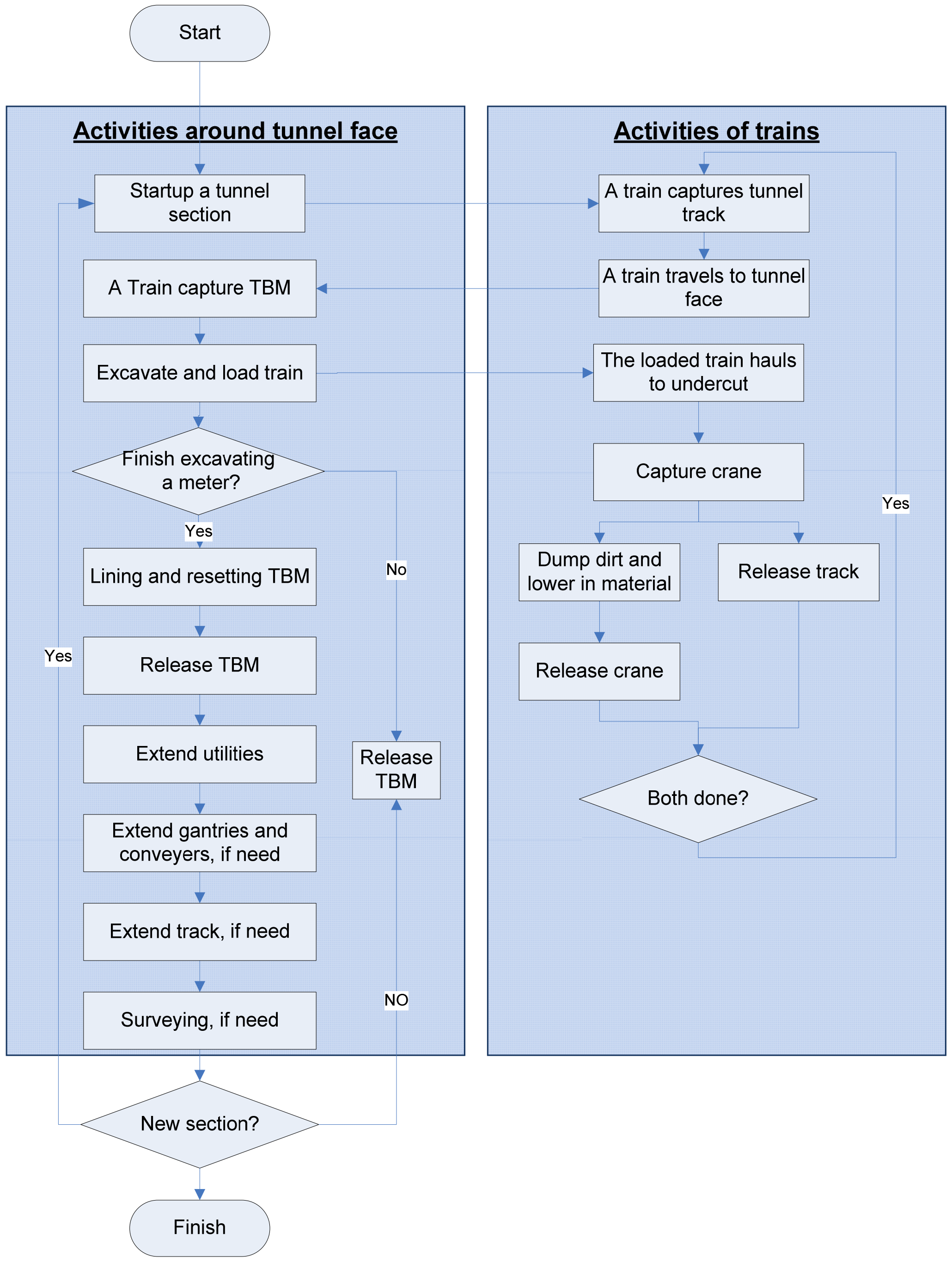


Figure 4-9 The flowchart of TBM tunnel construction (revised from AbouRizk (2010))

The simulation category simulates the tunnel construction process according to tunnel object property and its construction process. In order to simulate various utility tunnels, a universal utility tunnel object and its specifications are analyzed, and the simulation federates take into account the whole tunnel construction process, various resources used during construction, and different information from tunnel project stakeholders.

Federates under the special purpose category are designed for some specific functions. For example, the calendar and shift federate models the shift specification and controls shift on and off status. If there is no shift control in the federation, then the tunnel construction simulation federates just ignore shift status and construction duration will be measured in logical work hours only, instead of dates. Federates in this group are designed to add additional functions to the federation which could still work without special purpose federates.

The presentation category provides two types of reporting from simulation, including tunnel construction animation, and the planning federate for future tunnel construction. The planning federate is based on LPS.

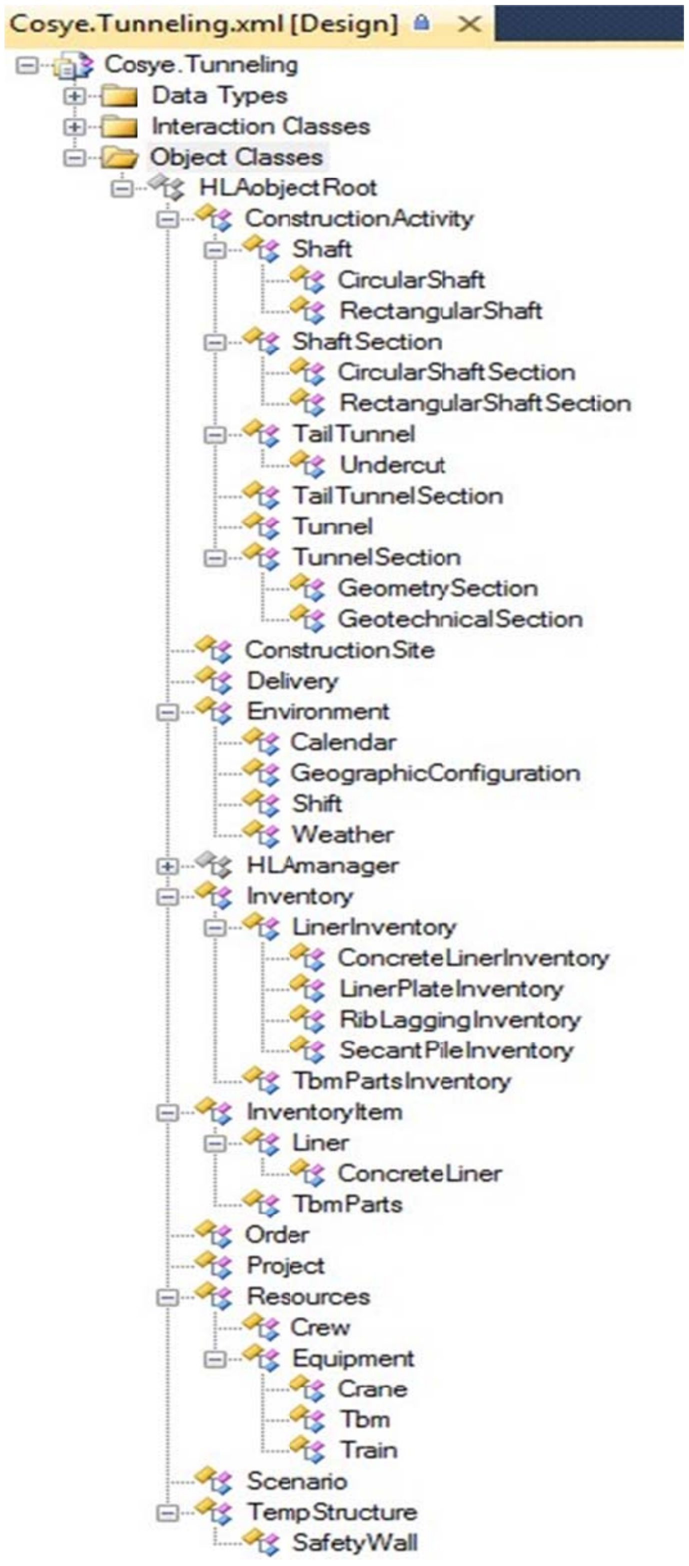
The function of each federate is described in Table 4-1. Among these federates, the shaft federate, tunnel federate, and dirt removal federate actually model the construction process using DES. The scenario setup federate defines the tunnel model specification for the other federates. This was implemented to increase usability by allowing users to specify model configurations and/or model inputs in one federate, so model developers do not need to change every element for model configuration. Additionally, the breakdown federate models machine breakdowns — the time to breakdown and time to repair of machine are analyzed and modeled. Some other supportive federates, such as the Calendar federate, were developed as a part of the tunnel simulation federation by the research group at the University of Alberta.

Table 4-1 Tunnel construction simulation module and its description

Federate	Function and Description
Shaft	Shaft construction process, including preparation, excavation, and lining
Tunnel	Simulate the advance process of TBM tunneling including excavation, lining and resetting; Simulate operational activities such as extend track, survey Simulate constructing undercut and tail tunnel Simulate tunnel start up operations including installing track/switcher, TBM, gantry and conveyer
Dirt Removal	Simulate dirt removal process in TBM tunneling Simulate crane operations of loading materials for both lining and unloading dirt from the bottom of a shaft to the ground
Scenario Setup	Define tunnel product model specification Define various resource specifications Specify initial status of the project
Breakdown	Calculate the time to breakdown, time to repair of machines Manage breakdowns by communicating with other federates through RTI

4.3.2 FOM

Federation object model (FOM) is used to describe the kinds of data, and their relationship, and these federates will be exchanged among federates in a federation execution. The FOM specifies the name and attribute(s) of each object class and each interaction class. Federates interact with the RTI according to FOM specification. Figure 4-10 shows most of the object classes in the FOM developed for this tunnel federation.

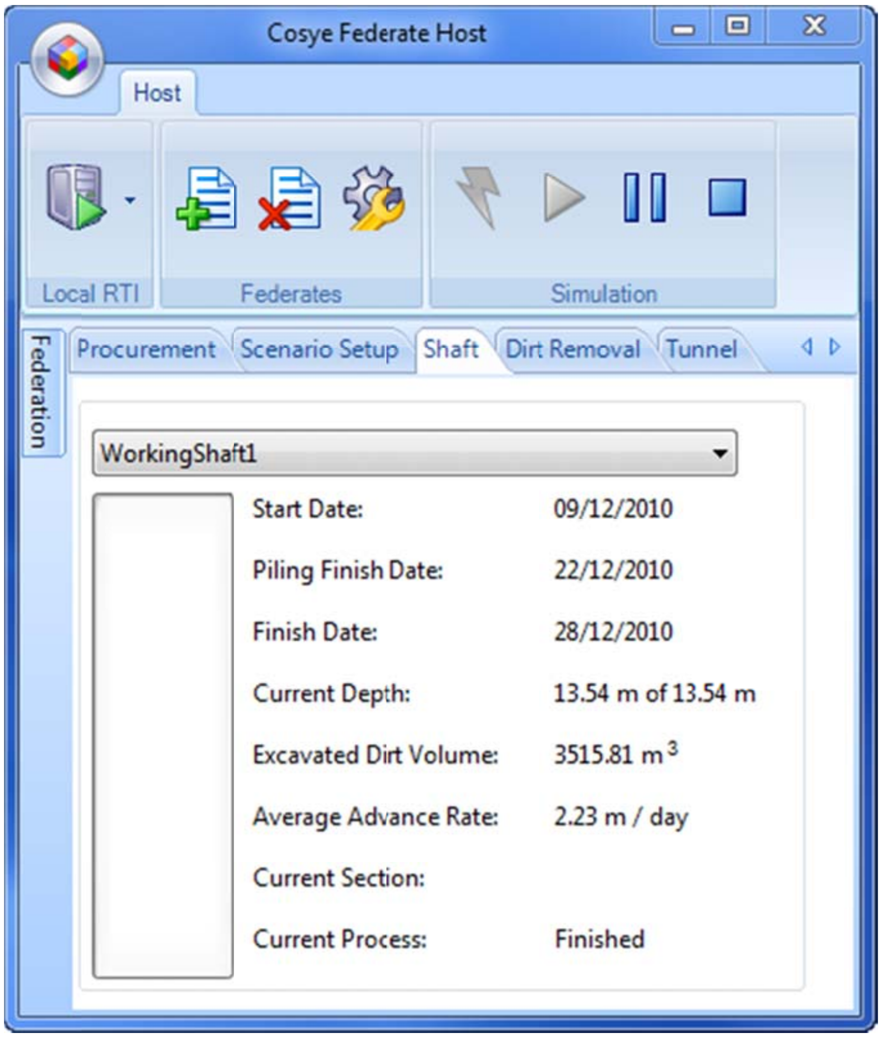


```
Cosye.Tunneling.xml [Design]
Cosye.Tunneling
    Data Types
    Interaction Classes
    Object Classes
        HLAobjectRoot
            ConstructionActivity
                Shaft
                    CircularShaft
                    RectangularShaft
                ShaftSection
                    CircularShaftSection
                    RectangularShaftSection
                TailTunnel
                    Undercut
                TailTunnelSection
                Tunnel
                TunnelSection
                    GeometrySection
                    GeotechnicalSection
            ConstructionSite
            Delivery
            Environment
                Calendar
                GeographicConfiguration
                Shift
                Weather
            HLAmanager
            Inventory
                LinerInventory
                    ConcreteLinerInventory
                    LinerPlateInventory
                    RibLaggingInventory
                    SecantPileInventory
                TbmPartsInventory
            InventoryItem
                Liner
                    ConcreteLiner
                TbmParts
            Order
            Project
            Resources
                Crew
                Equipment
                    Crane
                    Tbm
                    Train
            Scenario
            TempStructure
                SafetyWall
```

Figure 4-10 The object classes in the FOM in the tunnel federation

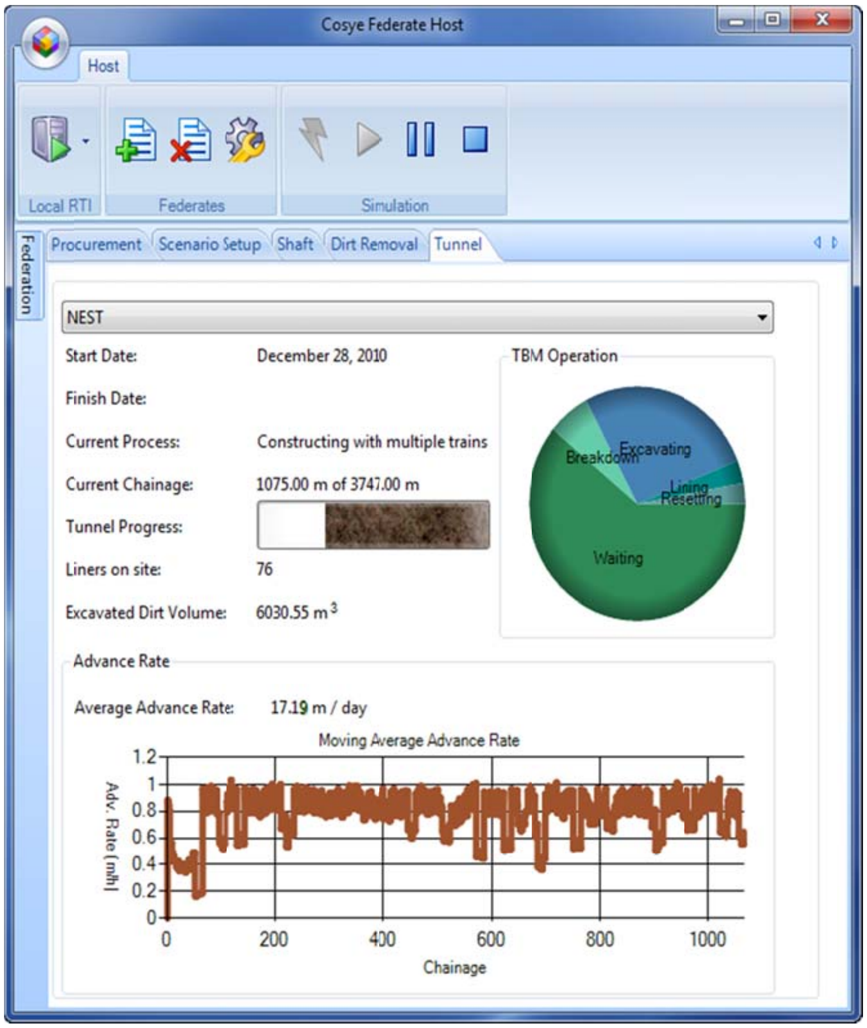
4.3.3 Tunnel construction simulation modeling

The tunnel construction process is modeled using DES, and implemented using CoSyE. CoSyE supports flexible development using multiple programming languages, such as C#, C++, Visual Basic etc., in MS Visual Studio. C# was chosen for developing a shaft federate, a tunnel federate, a dirt removal federate, and a scenario setup federate which together constitute this construction simulation module for tunneling. The interface of the shaft federate and tunnel federate, dirt removal federate, and breakdown federate are presented in the following Figures, and their codes are presented in the Appendix.



Start Date:	09/12/2010
Piling Finish Date:	22/12/2010
Finish Date:	28/12/2010
Current Depth:	13.54 m of 13.54 m
Excavated Dirt Volume:	3515.81 m^3
Average Advance Rate:	2.23 m / day
Current Section:	
Current Process:	Finished

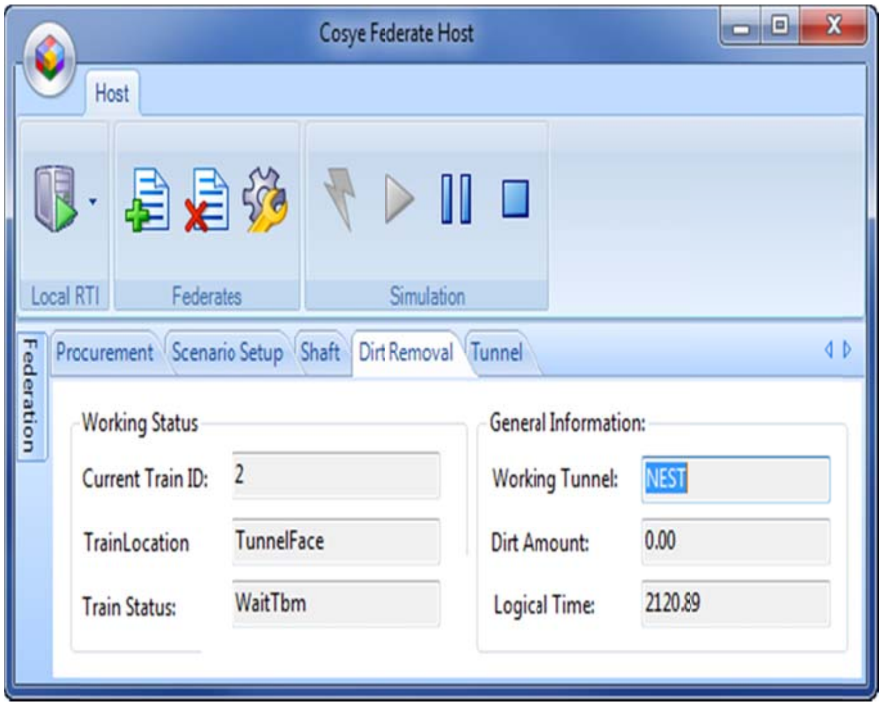
Figure 4-11 The interface of the shaft federate



Start Date: December 28, 2010
Finish Date:
Current Process: Constructing with multiple trains
Current Chainage: 1075.00 m of 3747.00 m
Tunnel Progress:
Liners on site: 76
Excavated Dirt Volume: 6030.55 m^3

Average Advance Rate: 17.19 m / day

Figure 4-12 The interface of the tunnel federate



Working Status

Current Train ID: 2

TrainLocation TunnelFace

Train Status: WaitTbm

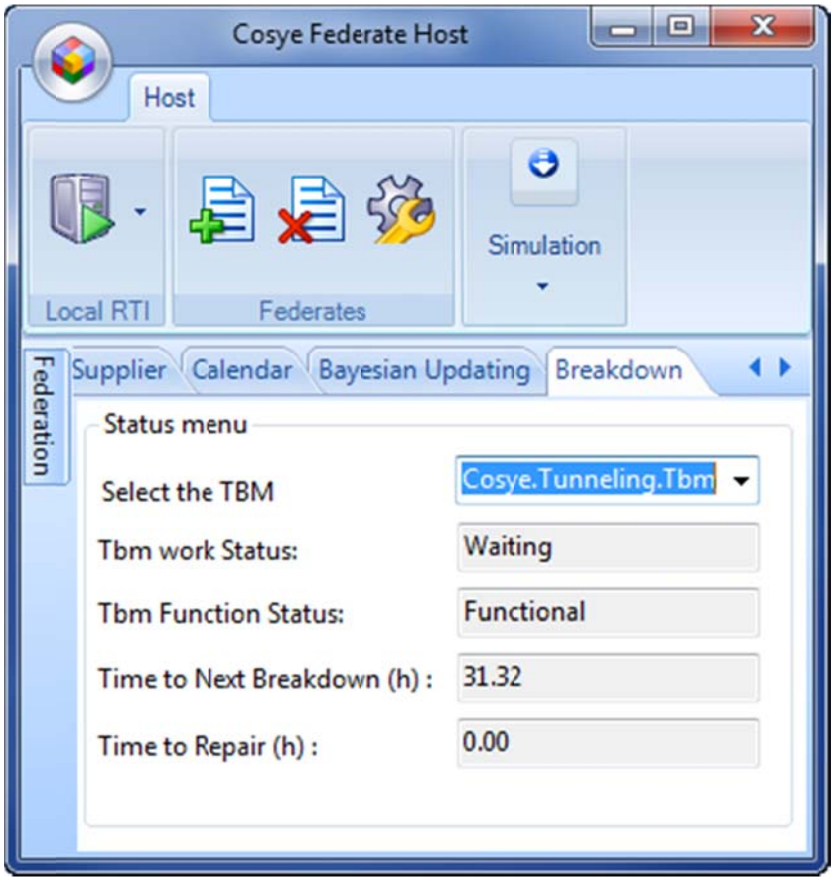
General Information:

Working Tunnel: NEST

Dirt Amount: 0.00

Logical Time: 2120.89

Figure 4-13 The interface of the dirt removal federate



Status menu

Select the TBM: Cosye.Tunneling.Tbm

Tbm work Status: Waiting

Tbm Function Status: Functional

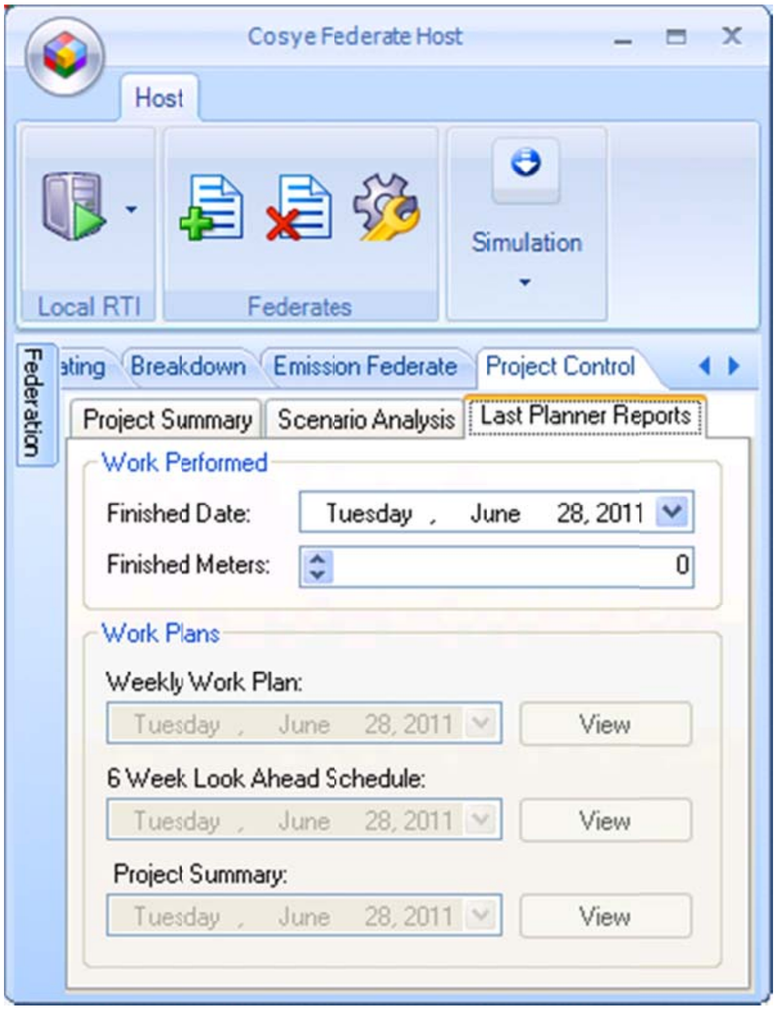
Time to Next Breakdown (h) : 31.32

Time to Repair (h) : 0.00

Figure 4-14 The interface of the breakdown federate

The interface of the reporting federate is shown in Figure 4-15. Users are allowed to select any work date and view the look-ahead plans based on that day. This module produces three types of plans and schedules at different levels of detail for project schedules and plans, including a project milestone plan, a six-week look-ahead schedule, and a weekly work plan, which correspond to initial planning, look-ahead planning, and commitment planning, as mentioned previously. The project milestone plan provides an overview of forecasted project performance.

The six-week look-ahead schedule and weekly work plan show the work plan for six weeks or a week into the future. Details and examples are provided in the case study.



Work Performed

Finished Date: Tuesday , June 28, 2011

Finished Meters: 0

Work Plans

Weekly Work Plan: Tuesday , June 28, 2011

6 Week Look Ahead Schedule: Tuesday , June 28, 2011

Project Summary: Tuesday , June 28, 2011

Figure 4-15 The interface of the reporting federate

The six-week look-ahead schedule and weekly work plan can present the schedule or plan of any future (six) week(s). The users just need to select the date of when the project schedules and plans should display. These plans and schedules are continuously updated throughout the construction process, based on results from simulation which model uncertainties and resource utilizations, and are kept up-to-date since the simulation model incorporates the latest project performance.

4.4 Application to a real tunnel construction project

4.4.1 Project introduction

An ongoing project, the North Edmonton Sanitary Trunk (NEST), taking place in Edmonton, Alberta, Canada, is selected for this case study. NEST represents a typical utility tunnel project, 3707 meters in length and 2.344 meters in diameter. The project began in December 2007 using a tunnel boring machine (TBM). The tunneling operations have been monitored every day and all data related to production progress, as well as all interruptions, have been well documented since construction began. Tunnel construction operations are characterized by their complexity and diversity, and are also usually linear and repetitive, so they have been selected for simulation modeling in this research.

4.4.2 The collaborative environment for construction projects

A construction project usually involves various project participants and diverse information in a distributed work environment, which makes data collection and sharing complex. A tunnel construction site involves numerous participants. Typical project participants in tunnel construction include labourers for construction work, suppliers who provide materials such as concrete liners, technicians who check and/or repair the tunnel boring machine (TBM), surveyors who make sure the tunnel alignment is correct, and so on. Different project participants have different concerns about a project and generate diverse types of data. Figure 4-16 shows a typical network of information sharing in a construction project. Successful project control depends on collaborative work from all project participants.

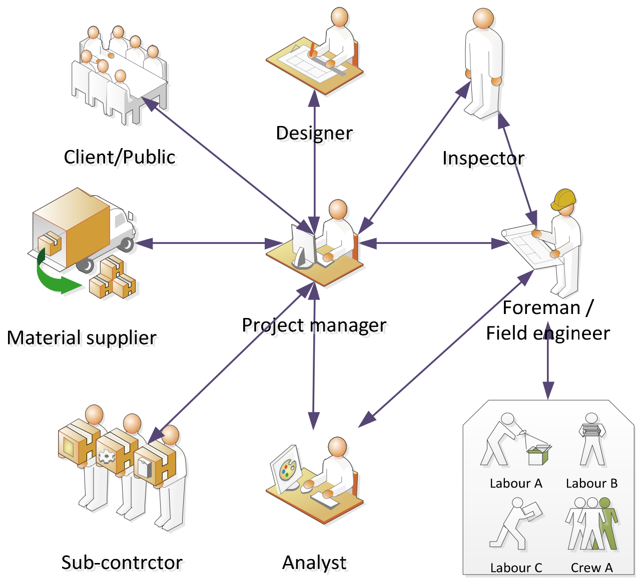


Figure 4-16 A typical information sharing network in a construction project

Project progress monitoring and control rely heavily on personal observation (visits) and analysis of a site, according to the tunnel construction practices of the City of Edmonton. Progress monitoring includes progress measurement, data collection, data analysis, reporting, etc. Various progress data must be collected, measured, and considered to support the decision making in project control. The data are usually distributed at locations such as construction sites and offices. Although collected information could be transferred using computers and the Internet/intranet, the data collection for project monitoring in the construction domain still heavily relies on personal site visiting, because the project progress should be assessed by human judgment.

Data collection is sometimes undertaken simultaneously by different project participants for different purposes; for example, in the case study project, two similar processes of site data collection were identified. The information flow generated by a consulting party includes weekly site visits for progress monitoring and monthly reporting for progress summary and analysis. It provides project managers with a summary of all previous project performance. Meanwhile, the City has an in-house crew to monitor different project job sites and prepare daily progress reports describing the performance of a single workday for each job site, as shown in Figure 4-17. Huge time and effort overlap and waste have been identified in collecting and transferring site information to the head office for project managers. Moreover, a lengthy delay in report updating is another issue that hinders high-level managers in identifying problems.

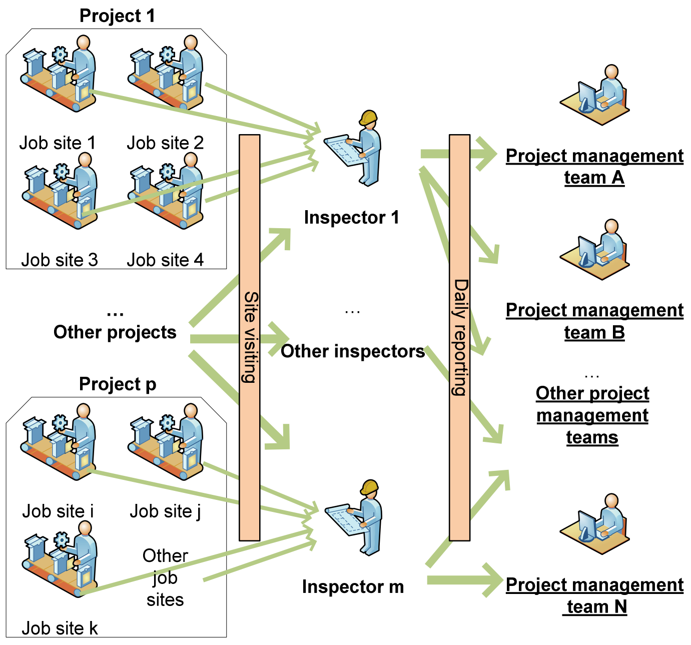


Figure 4-17 Current information flow for project control in City of Edmonton

The improved information flow between project stakeholders is depicted in Figure 4-18. It also demonstrates how this simulation system can be used in real construction projects to enhance project control for various project stakeholders during the construction process. The cylinder represents the progress monitoring and control module, and the tunnel federation contains the other modules as described in previous sections, including the Bayesian updating module, the construction simulation module, and the scheduling and planning module. First, the progress monitoring and control module allows different project stakeholders to access progress information, and provides automated information collection and sharing to improve project management. It supports the involvement of various project participants for data collection, storing, analysis and review; for example, inspectors for as-built data of project progress data, designers for tunnel specifications, suppliers for material delivery information, and project managers for varied information. Then, collected progress data is analyzed for updating simulation model inputs, which continuously tune the simulation model to produce better simulation along with project progress and changes. The simulation model then produces look-ahead work plans. The overall process enables the tunnel federation to produce more accurate project plans using the latest progress data.

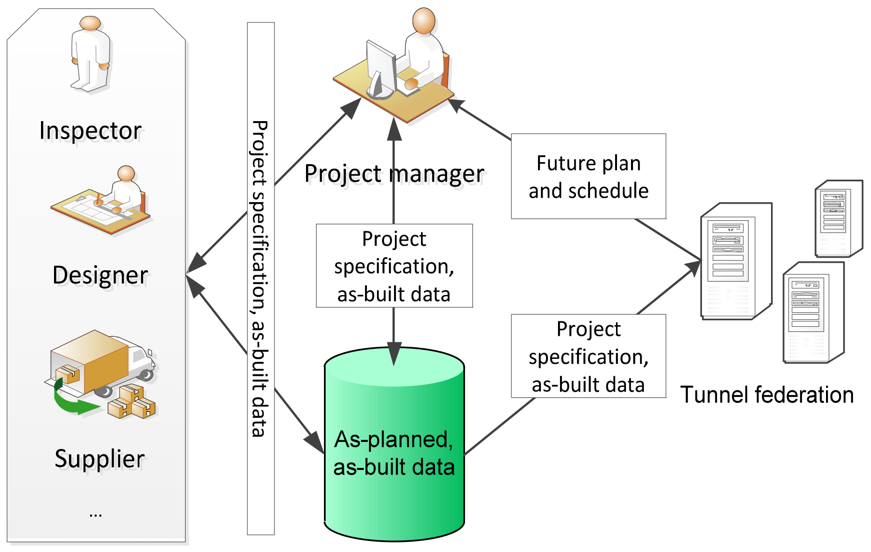

Inspector
Designer
Supplier
...
Project specification, as-built data
Project manager
Project specification, as-built data
Future plan and schedule
Project specification, as-built data
As-planned, as-built data
Tunnel federation

Figure 4-18 Improved information flow using the simulation system

4.4.3 Modeling

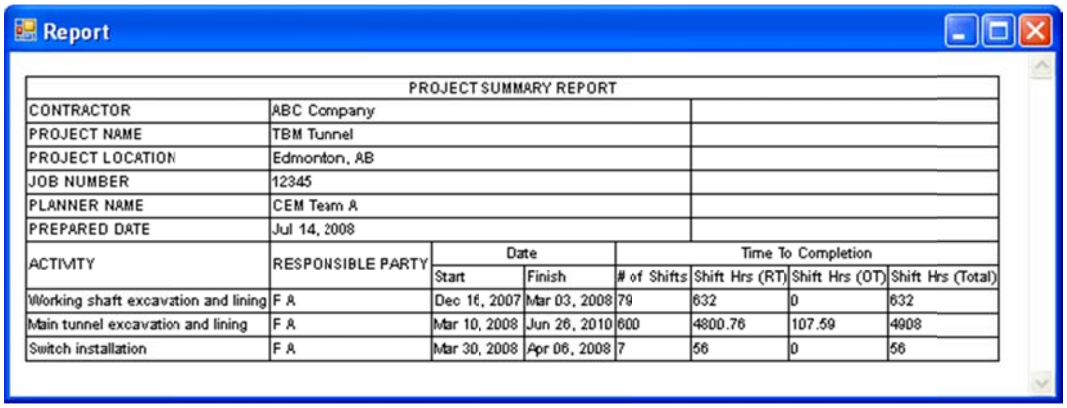
The tunnel construction process in the case study project was modeled using discrete event simulation, according to the construction process and project logic, which was analyzed by consulting project managers and field engineers, reading technical documents, and visiting construction sites. The simulation model accounted for resource constraints, material procurement and supply, and project logic.

The Bayesian inference was then applied to update key parameters for the simulation model. TBM penetration and machine breakdown were chosen as key model inputs for updating in this case study. According to Xie et al. (2011), a Gamma distribution showed improved goodness of fitness over a Normal distribution for the TBM penetration rate. Thus, a Gamma distribution was chosen as a conjugate prior in TBM penetration rate updating. Gamma/Exponential distributions have been applied for modeling equipment failure rates in many studies (NIST/SEMATECH 2003; Shafaghi 2008; Zubair et al. 2011). An Exponential distribution is a special type of Gamma distribution with a shape parameter of 1. Both Exponential distributions and Gamma distributions are considered members of the exponential family in statistics.

TBM breakdown data were categorized into major breakdowns and minor breakdowns. Major breakdowns were usually due to bad geotechnical conditions. For example, hard rock in soil can break the rotating head and it takes more than a day to fix. Minor breakdowns were usually less than a standard shift length, which is 8 hours. Bayesian inference was applied for both major and minor breakdowns.

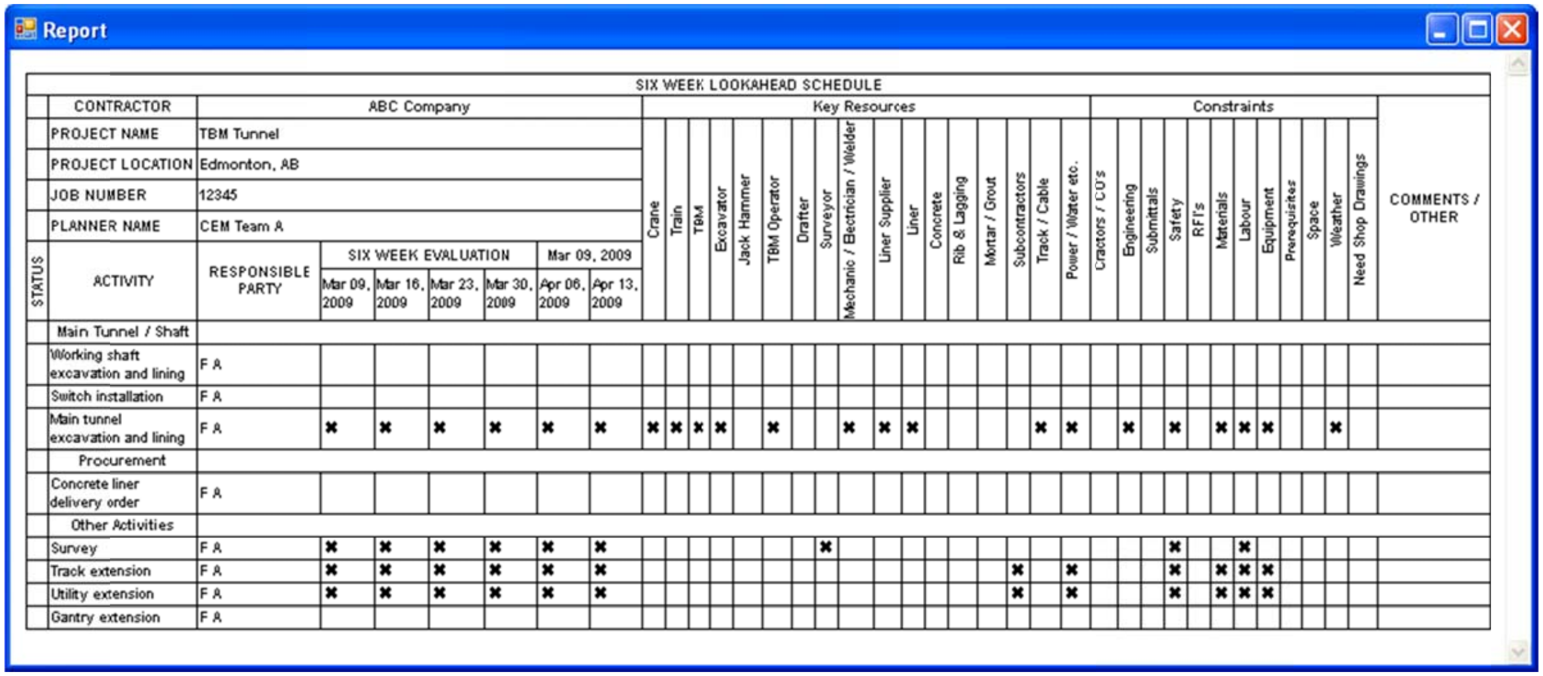
4.4.4 Planning

Three different plans and schedules are produced using updated model inputs and as-built project data. The project milestone plan shows the time and crew hours required per major project task, and reflects the master schedule, as shown in Figure 4-19. The six-week look-ahead schedule, as shown in Figure 4-20 summarizes the main activities which will carry on within the next six weeks, with their corresponding resources and constraints of all ongoing tasks. This schedule provides the trend of an ongoing project and can help highlight the underlying project logic. The weekly work plan specifies a week’s project activities and their progress, which may be quantified according to the simulation. An example is presented in Figure 4-21, showing the planned progress of main tunnel construction in the NEST project.



PROJECT SUMMARY REPORT							
CONTRACTOR	ABC Company						
PROJECT NAME	TBM Tunnel						
PROJECT LOCATION	Edmonton, AB						
JOB NUMBER	12345						
PLANNER NAME	CEM Team A						
PREPARED DATE	Jul 14, 2008						
ACTIVITY	RESPONSIBLE PARTY	Date		Time To Completion			
		Start	Finish	# of Shifts	Shift Hrs (RT)	Shift Hrs (OT)	Shift Hrs (Total)
Working shaft excavation and lining	F A	Dec 16, 2007	Mar 03, 2008	79	632	0	632
Main tunnel excavation and lining	F A	Mar 10, 2008	Jun 26, 2010	600	4800.76	107.59	4908
Switch installation	F A	Mar 30, 2008	Apr 06, 2008	7	56	0	56

Figure 4-19 Project milestone plan from simulation system



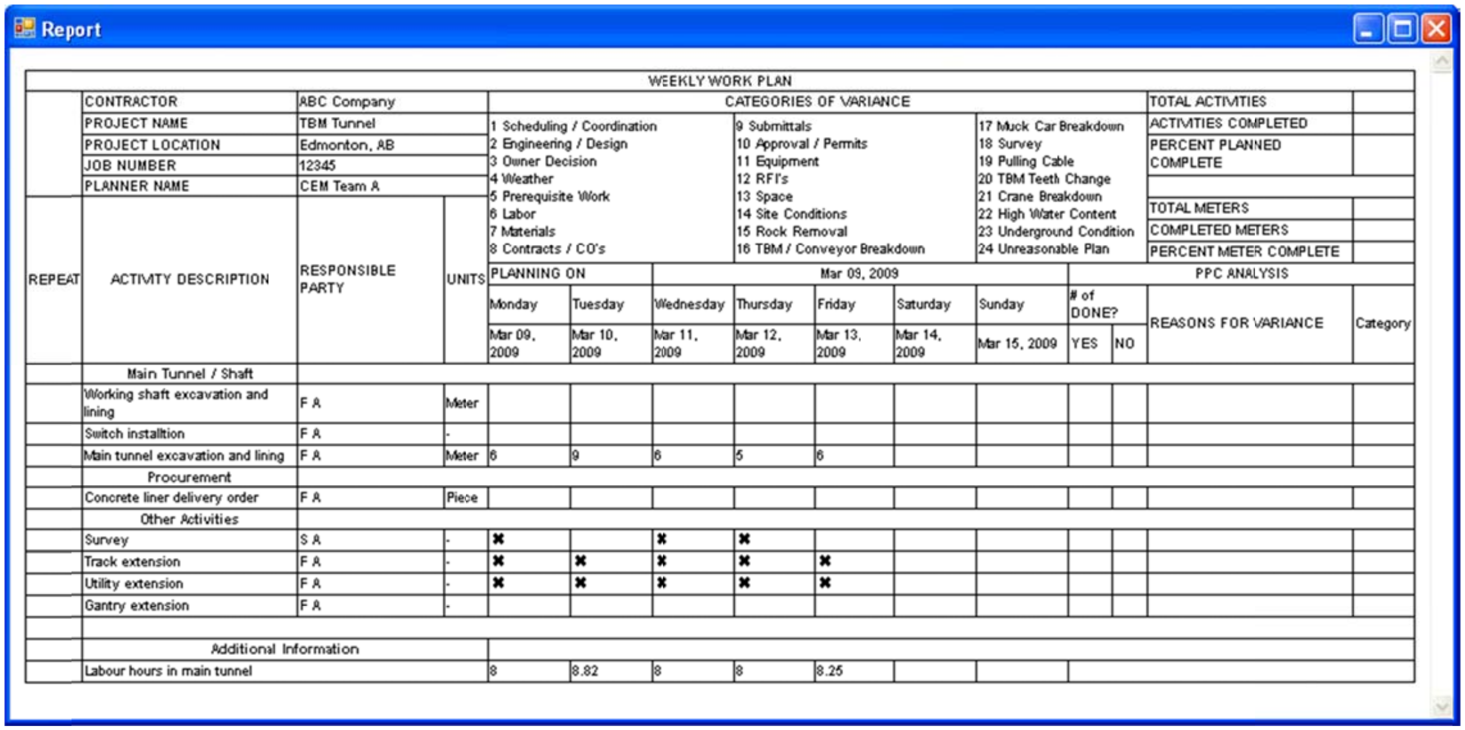
Report

SIX WEEK LOOKAHEAD SCHEDULE

CONTRACTOR	ABC Company
PROJECT NAME	TBM Tunnel
PROJECT LOCATION	Edmonton, AB
JOB NUMBER	12345
PLANNER NAME	CEM Team A

STATUS	ACTIVITY	RESPONSIBLE PARTY	SIX WEEK EVALUATION				Mar 09, 2009		Key Resources																	Constraints												COMMENTS / OTHER
			Mar 09, 2009	Mar 16, 2009	Mar 23, 2009	Mar 30, 2009	Apr 06, 2009	Apr 13, 2009	Crane	Train	TBM	Excavator	Jack Hammer	TBM Operator	Drafter	Surveyor	Mechanic / Electrician / Welder	Liner Supplier	Liner	Concrete	Rib & Lagging	Mortar / Grout	Subcontractors	Track / Cable	Power / Water etc.	Cractors / CU's	Engineering	Submittals	Safety	RFI's	Materials	Labour	Equipment	Prerequisites	Space	Weather	Need Shop Drawings	
	Main Tunnel / Shaft																																					
	Working shaft excavation and lining	F A																																				
	Switch installation	F A																																				
	Main tunnel excavation and lining	F A	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖		✖			✖	✖	✖					✖	✖		✖		✖		✖	✖	✖			✖		
	Procurement																																					
	Concrete liner delivery order	F A																																				
	Other Activities																																					
	Survey	F A	✖	✖	✖	✖	✖	✖								✖													✖			✖						
	Track extension	F A	✖	✖	✖	✖	✖	✖															✖		✖				✖		✖	✖	✖					
	Utility extension	F A	✖	✖	✖	✖	✖	✖															✖		✖				✖		✖	✖	✖					
	Gantry extension	F A																																				

Figure 4-20 An example of a six-week look-ahead schedule from simulation system



Report

WEEKLY WORK PLAN

		CATEGORIES OF VARIANCE				
CONTRACTOR	ABC Company				TOTAL ACTIVITIES	
PROJECT NAME	TBM Tunnel	1 Scheduling / Coordination	9 Submittals	17 Muck Car Breakdown	ACTIVITIES COMPLETED	
PROJECT LOCATION	Edmonton, AB	2 Engineering / Design	10 Approval / Permits	18 Survey	PERCENT PLANNED COMPLETE	
JOB NUMBER	12345	3 Owner Decision	11 Equipment	19 Pulling Cable		
PLANNER NAME	CEM Team A	4 Weather	12 RFI's	20 TBM Teeth Change		
		5 Prerequisite Work	13 Space	21 Crane Breakdown		
		6 Labor	14 Site Conditions	22 High Water Content	TOTAL METERS	
		7 Materials	15 Rock Removal	23 Underground Condition	COMPLETED METERS	
		8 Contracts / CO's	16 TBM / Conveyor Breakdown	24 Unreasonable Plan	PERCENT METER COMPLETE	

REPEAT	ACTIVITY DESCRIPTION	RESPONSIBLE PARTY	UNITS	PLANNING ON	Mar 09, 2009						PPC ANALYSIS			
				Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	# of DONE?		REASONS FOR VARIANCE	Category
				Mar 09, 2009	Mar 10, 2009	Mar 11, 2009	Mar 12, 2009	Mar 13, 2009	Mar 14, 2009	Mar 15, 2009	YES	NO		
	Main Tunnel / Shaft													
	Working shaft excavation and lining	F A	Meter											
	Switch installtion	F A	-											
	Main tunnel excavation and lining	F A	Meter	6	9	6	5	6						
	Procurement													
	Concrete liner delivery order	F A	Piece											
	Other Activities													
	Survey	S A	-	✖		✖	✖							
	Track extension	F A	-	✖	✖	✖	✖	✖						
	Utility extension	F A	-	✖	✖	✖	✖	✖						
	Gantry extension	F A	-											
	Additional Information													
	Labour hours in main tunnel			8	8.82	8	8	8.25						

Figure 4-21 An example of weekly work plan from simulation system

The labour hours are also presented in the weekly work plan and the milestone plan. Although in many other construction projects, everyday shift lengths or work hours are fixed, a tunnel construction site usually has variable daily work hours due to the unique property of tunneling. If the construction crew is in the process of excavating or lining a tunnel when specified work hours are reached, the crew will continue work until the lining is finished. A tunnel should be properly supported by the end of a shift before all crew finish work and leave the construction site, unless another crew will immediately continue work on it. Therefore, a tunneling construction crew often works longer than the specified work hours. The every day labour hours are shown in the weekly work plan, and the total labour hours are summarized in the milestone plan, and classified into regular time and overtime.

The key resources and constraints of each activity are predefined in the system. The author consulted project managers from the City of Edmonton and identified a list of resources and constraints related to each activity. If an activity will be performed, then the corresponding resources and constraints are selected in the six-week look-ahead schedule. These predefined relationships served as a knowledge base for scheduling and planning and could be updated throughout project execution. In addition, both the six-week look-ahead schedule and the weekly work plan specify the responsible party of each task, which is called the last planner, according to LPS.

The produced six-week look-ahead schedule and weekly work plan can be used by companies which are using LPS for project control. Usually, LPS plans are produced through conversations among last planners and project managers who reach a consensus on future project progress. The tasks and their corresponding progress in the six-week look-ahead schedule and the weekly work plan can be interpreted as commitments. However, these conversations require substantial time and effort for communication to gain consensus. They may need further confirmation from last planners and project managers, because the actual production and project performance are affected by many factors, such as the workload of each last planner in the real world. But these schedules and plans can be used as proposed base plans for achieving the success identified in the conversations, and helping project managers to understand and guide the overall project.

4.5 Discussion

4.5.1 Improvement in tunnel modeling

This HLA-based tunnel simulation system enables modeling in a collaborative environment. It allows building of the simulation model from simulation components, integrating data collection, input updating, simulation modeling and planning. Most current construction simulation tools only allow the development and run of a simulation model on a standalone computer in a specific environment. This means that a model developer must collect data from various sources and set up all model inputs, and special expertise and collaborative work among different project participants is required to setup and run a model.

This simulation system demonstrated how to integrate construction information collection, data analysis and updating, discrete event simulation modeling, and planning in a collaborative way. It also improves the interoperability and reusability of simulation modeling, as well as dynamic and collaborative project control.

This model also showed improvement in modeling shift length in a more reasonable way. Shift length is usually predefined (e.g. 8 hours) in a simulation model. The actual shift length in a tunnel construction site is usually not exactly 8 hours. For example, if the current task is lining using TBM, at the end of a shift, the crew should finish lining before they leave a tunnel site for safety purposes, because the tunnel could collapse wihtout proper support. However, this logic is usually ignored in previous tunnel simulation models, but is now taken into account. The developed simulation model terminates a shift after an event is completed. If the task at the end of a base shift length is excavating, resetting, or lining, this simulation model will continue running until the completion of lining task. Previous simulation models allow one event to span two shifts to make the shift length match predefined base shift length, which should not be.

This simulation model models machine breakdown in a more reasonable way compared to previous simulation models. Breakdown is usually modeled as an event in discrete simulation model, and is scheduled to happen as a queued event. In other words, it can only happen between two activities. This simulation model allows breakdown to happen at any designated time, by preempting the machine, and pausing the event. In addition, this research applied Bayesian updating technique to update the mean time between failure and mean time to repair for TBM.

4.5.2 Look-ahead planning

A look-ahead planning (LAP) meeting is used to establish look-ahead plans (Macomber, 2004). Project performers attend this meeting to prepare requests for the work to be performed in the coming weeks. The performers look for the conditions of the coming work and identify the constraints for making the work executable.

LAP meetings achieve the following objectives (Macomber, 2004): establish a look-ahead plan, identify the work constraints, secure the promises for addressing the constraints, and launch the coming work tasks to the performers.

This research produced the base plan for look-ahead planning through a simulation model. In order to establish reasonable plans, the simulation model took the resource requirement and construction logic constraints into account. Although the produced plan can not secure the promises from various project performers, this method definitely saves time and effort for the work performers to establish the basis for reliable look-ahead planning.

The project plans are established in a systematic way through simulation modeling. This is considered an improvement compared with other simulation models, which only provide information of a summary of time and cost to completion. It helps systematic and dynamic planning by presenting a milestone report, a six-week look-ahead schedule, and a weekly work plan. These plans provide model details of the ongoing project, and help users gain insights into the project.

4.5.3 Limitation

One limitation of this research is that it focused on tunnel construction, and is limited to TBM tunneling. There are many different tunnel construction methods, such as sequential excavation, that could be investigated and modeled. The current simulation model is not capable of simulating various tunneling methods and provides more flexibility and capability in what-if scenario analysis.

This simulation system supports limited decision variables for scenario analysis, including shift length, tunnel length, the number of muck carts, and the capacity of each cart. This system could be further developed to allow more decision variables for assisting a project management team in decision-making.

As the simulation system is hard coded, users have to change the code to try different scenarios. A well-designed interface could make it user friendly to support decision-making, and could easily incorporate the expertise and ideas from project management professionals, who do not usually have a background in computer programming.

For further improvement of this approach, the author recommends integrating automated data collection technology, such as RFID, to measure and collect project progress data, and investigating new techniques in updating model inputs other than Bayesian inference, such as time series, for better input updating.

4.6 Summary

The developed system utilized HLA to facilitate collaborative project control among project participants with different project interests and project roles, including construction teams, project managers, inspectors, etc. It can be used for what-if scenario analysis and assisting decision-making. The system can provide different plans and schedules according to different model setups and model parameters such as the number of muck cars, the capacity of each muck car, and crane productivity. These plans and schedules, which not only present project schedules and costs, but also identify the resource constraints, can be compared and analyzed to contribute to further decision-making among project stakeholders. The proposed system has been demonstrated and implemented for a tunnel construction project in the case study of this paper. However, the methodology and the system architecture are generic and can be applied in many different projects regardless of project type.

CHAPTER 5 ENHANCING QUALITY OF TBM PENETRATION RATE FORECAST USING ACTUAL PROJECT INFORMATION: BAYESIAN UPDATING

5.1 Introduction

Maintaining valid model inputs is a challenge for long-term use of simulation models. When developing a discrete simulation model for a project which has not started, model inputs are usually assumed. The assumptions are based upon historical data from similar projects and similar activities, and from experts' judgments. As projects progress and project circumstances change, simulation model inputs need to be updated to reflect these changes, so as to maintain model validity.

The Bayesian approach provides a method for updating model parameters by combining both historical data and new available information. Bayes' inference can be interpreted as $posterior \propto likelihood \times prior,$ where posterior, likelihood, and prior can be interpreted as the posterior distribution, likelihood distribution, and prior distribution of any model inputs in this research. As stated earlier, before a project starts, assumptions of model inputs are made based on expert judgment and/or historical data from similar projects. The initial assumption of duration is assumed to be continuous and is represented by a probability density function (PDF) in the simulation model. These initial PDFs are the prior distribution. As a project progresses and the actual project performance is observed, and the project uncertainty reduced, the original assumption should be revised according to the observed data. The new available information is the likelihood distribution, and the revised PDF is the posterior distribution. This study uses the penetration rate of the TBM as an example to illustrate how progress data is utilized and model input is calculated in simulation.

5.2 Bayesian Inference

5.2.1 Bayes' theorem

The Bayesian approach is built upon a statistical proposition, Bayes' theorem, as described in Formula (5-1):

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{5-1}$$

Where, $P(A)$ and $P(B)$ are the marginal probability densities of event A and B, $P(A|B)$ and $P(B|A)$ are the conditional densities, if A and B stand for a pair of random variables: of x and y, respectively. The Formula (5-1) is written as

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{5-2}$$

Where $p(x)$ and $p(y)$ are the marginal probability densities of x and y; $p(x|y)$ and $p(y|x)$ are the conditional densities. The denominator, $p(y)$, in (5-2), the marginal distribution of y, is considered as a normalization constant. It does not depend on θ, and is a constant with fixed y. In the application of Bayes' theorem, sometimes y represents the observed data and is denoted by y^{obs}, while x means the parameters of the model, and is denoted by θ. Thus, a new form of the Bayes' theorem is presented as follows:

$$\pi(\theta|y^{obs}) \propto p(y^{obs}|\theta)\pi(\theta) \tag{5-3}$$

Equation (5-3) is also called *unnormalized posterior density*. When $p(y^{obs}|\theta)$ is considered as a function of θ, it is called the likelihood function, and we write $l(\theta|y^{obs}) = p(y^{obs}|\theta)$. Then we have

$$\pi(\theta|y^{obs}) \propto l(\theta|y^{obs})\pi(\theta) \tag{5-4}$$

This equation is interpreted as $posterior \propto likelihood \times prior$. $\pi(\theta)$ is called prior distribution because it is a priori belief about the values of θ. The uncertainty of the unknown parameters θ is captured by the observations, y^{obs}. Here Bayes' theorem provides a way to incorporate the observed information and update the parameter uncertainty. $\pi(\theta|y^{obs})$ is then called posterior distribution, representing the updated uncertainty of prior distribution.

The marginal distribution of random variable y is written as

$$\pi(y) = \int p(y|\theta)\pi(\theta)\,d\theta \tag{5-5}$$

Which is called the *prior predictive distribution* of Y, because it represents the current predictions of the value of Y, incorporating both the uncertainty about the value of θ and the conditional uncertainty about y when θ is known. If Y is a time-ordered random variable, the observations up to time t is expressed as $y_t^{obs} = (y_0, y_1, y_2, \ldots, y_t)$, and the observation y_{t+1} can be predicted as follows:

$$\pi(y_{t+1}|y_t^{obs}) = \int p(y_{t+1}|y_t^{obs}, \theta)\pi(\theta|y_t^{obs})d\theta \tag{5-6}$$

$\pi\left(y_{t+1}|y_t^{obs}\right)$ is called the *posterior predictive distribution* and is usually used for prediction.

5.2.2 Conjugate distributions

Difficulties arise when implementing Bayesian approach. Multidimensional integration is involved in finding the normalizing constant for Equation 5-3 and the predictive distribution in Equation 5-6. The key problem lies in finding analytical solutions for the integral.

Two solutions have been sought for the integration problems. Before the pervasiveness of cheap and powerful computers, the research focus was finding pairs of conjugate distributions. When the prior distribution type is the same as posterior distribution type, the prior and posterior are called conjugate distributions. Conjugate distributions provide convenience in calculating the tractable analytical solutions to the integral. The other approach is sampling-based estimation. A sufficient number of samples from a density can approximate the mathematical form of that density through curve estimation methods. For example, a variety of Markov Chain Monte Carlo (MCMC) simulation has been widely used in sampling from posterior densities to seek approximate solutions for Bayesian inference. Conjugate distributions are applied in this research. The following introduces two pairs of conjugate distributions implemented in the research.

5.2.2.1 Conjugate prior – Normal distribution

Normal distribution is the conjugate prior of a normal likelihood distribution. The posterior is normally distributed. The parameters of posterior, $\pi(\theta\big|m', v')$, can be obtained by

$$\begin{cases} m' = \dfrac{\sigma^2 m + vn\bar{x}}{\sigma^2 + vn} \\ v' = \dfrac{v\sigma^2}{\sigma^2 + vn} \end{cases} \tag{5-7}$$

Where,

m', m - the mean of posterior and prior Normal distribution respectively;

$\bar{x}$ - the mean of observed data;

v', v - the standard deviation of posterior and prior distribution respectively;

σ – the standard deviation of observed data.

5.2.2.2 Conjugate prior – Gamma distribution

Gamma distribution is the conjugate prior of an exponential likelihood distribution. The posterior obeys Gamma distribution. The parameters of posterior, $\pi(\theta\big|\alpha', \beta')$, can be obtained by

$$\begin{cases} \alpha' = \alpha + n \\ \beta' = \dfrac{\beta}{1 + \beta \sum_{i=1}^{n} x_i} \end{cases} \tag{5-8}$$

Where,

α', α - the shape parameters of posterior and prior Gamma distribution respectively;

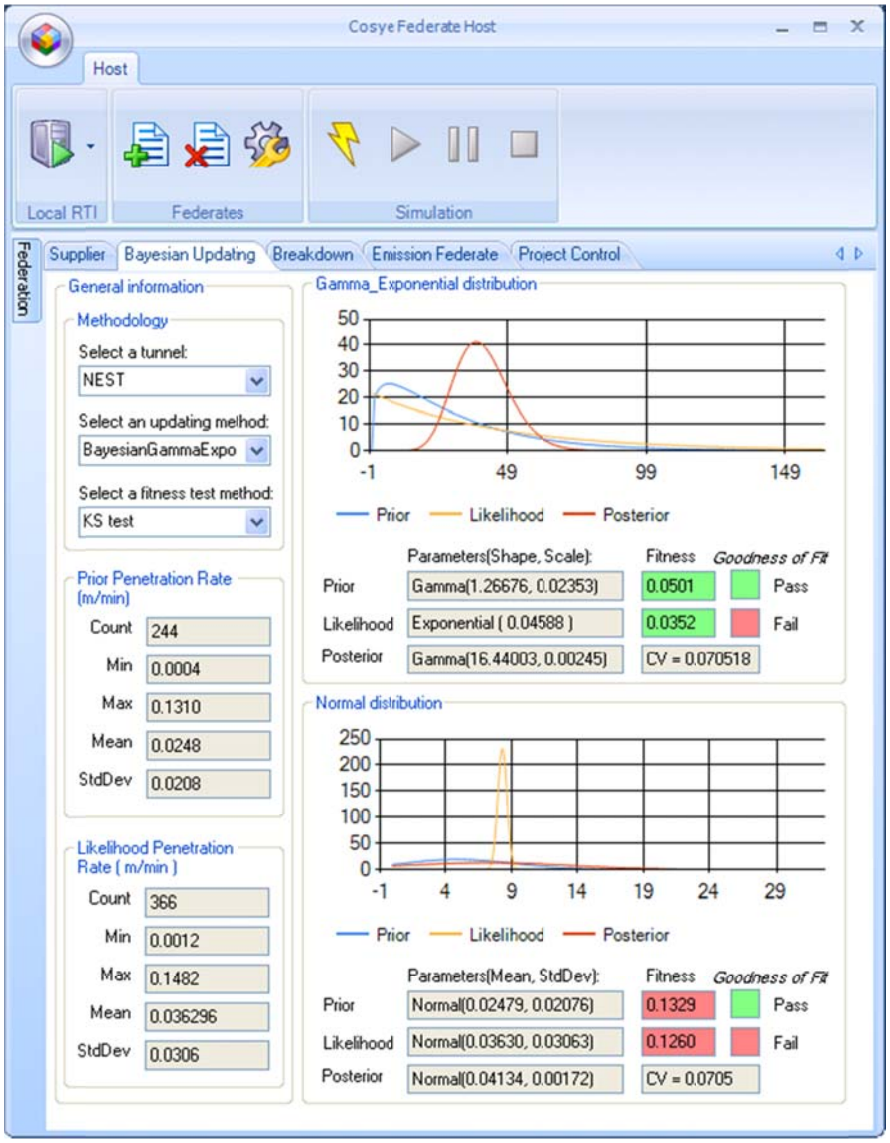
β', β - the scale parameters of posterior and prior Gamma distribution respectively;

n - the number of observed data or sample size;

x_i - the value of the ith of observed data.

5.3 Bayesian updating federate

The Bayesian updating module is a self-contained simulator, called the Bayesian updating federate, which interacts with the other simulation components as needed. It was developed in CoSyE, as shown in Figure 5-1. It combines both historical data and new available information for simulation model inputs. The observed data increases when construction continues, so the likelihood data size also grows. The posterior is updated consecutively according to newly collected data until project completion, in order to reflect the latest changes in project circumstances. The Bayesian updating federate incorporates new information from the progress monitoring module and automatically updates the simulation model, working in conjunction with the progress monitoring module to continuously and promptly update inputs in the simulation module to ensure model validity.



General information

Methodology

Select a tunnel: NEST

Select an updating method: BayesianGammaExpo

Select a fitness test method: KS test

Prior Penetration Rate (m/min)

Count	244
Min	0.0004
Max	0.1310
Mean	0.0248
StdDev	0.0208

Likelihood Penetration Rate (m/min)

Count	366
Min	0.0012
Max	0.1482
Mean	0.036296
StdDev	0.0306

Gamma_Exponential distribution

	Parameters(Shape, Scale):	Fitness	Goodness of Fit	
Prior	Gamma(1.26676, 0.02353)	0.0501	🟩	Pass
Likelihood	Exponential (0.04588)	0.0352	🟥	Fail
Posterior	Gamma(16.44003, 0.00245)	CV = 0.070518		

Normal distribution

	Parameters(Mean, StdDev):	Fitness	Goodness of Fit	
Prior	Normal(0.02479, 0.02076)	0.1329	🟩	Pass
Likelihood	Normal(0.03630, 0.03063)	0.1260	🟥	Fail
Posterior	Normal(0.04134, 0.00172)	CV = 0.0705		

Figure 5-1 Bayesian updating federate

This federate provides two pairs of conjugate distributions when updating model inputs. When the prior distribution type is the same as the posterior distribution type, the prior and posterior are called conjugate distributions. More details of conjugate distributions and their calculations are presented later in this Chapter. In the first conjugate distribution, prior, likelihood, and posterior are assumed to be normally distributed. This method was implemented by Chung et al. (2006) to predict tunnel boring machine penetration rates and project productivity. Chung et al. (2006) showed that the prediction approaches actual performance as the project progresses and more actual performance data is used. The second method assumes a Gamma distribution for prior and posterior and an exponential distribution for likelihood.

The goodness of fitness of both prior and likelihood are provided based on Kolmogorov-Smirnov (K-S) test, as shown in Figure 5-1. When applying Gamma distribution as prior, the K-S test statistic is 0.0501. When applying Exponential distribution as likelihood, the K-S test statistic is 0.0352. They are less than the critical value, and shown in green background in the Figure. Thus, they passed the K-S test. When applying Normal distribution as prior, the K-S test statistic is 0.1329. When applying Normal distribution as likelihood, the K-S test statistic is 0.1260. They are less than the critical value, and shown in red background in the Figure. Thus, they failed the K-S test. Therefore, Gamma distribution represents the data set of TBM penetration rate better when compared to Normal distribution.

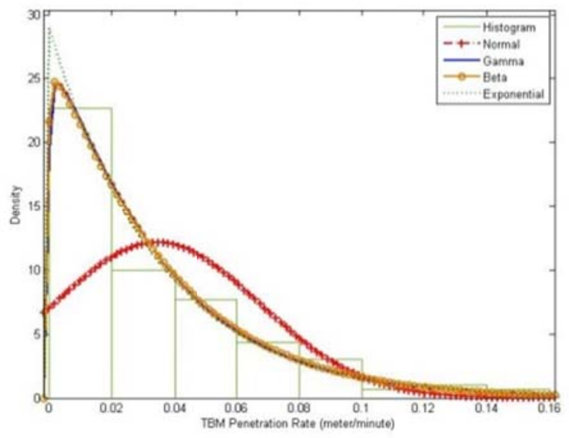
5.4 TBM penetration rate updating and analysis

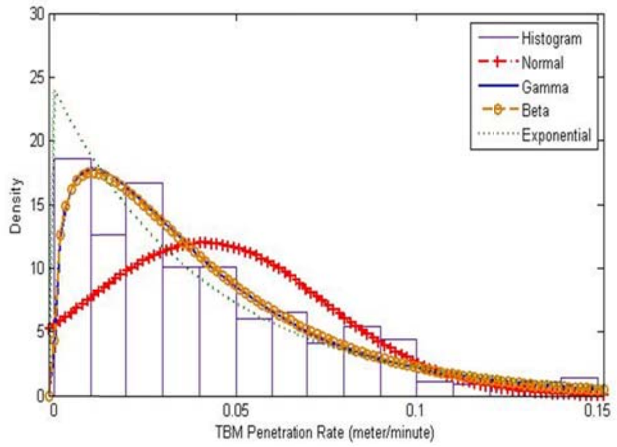
This Bayesian updating federate is used for updating inputs for simulation models. Simulation model inputs are based on historical data and site observations. TBM is the key resource in TBM tunneling. Its production efficiency is defined by penetration rate, representing the excavating distance per unit time. Ourdev, Abourizk, and Al-Bataineh (2007) proposed a way of inferring TBM penetration rate based on tunnel progress data. This study followed their method, and it is reviewed here briefly.

TBM penetration rate is calculated as the daily production divided by the TBM's corresponding production time. The production time equals the total time needed to complete 1 meter of tunnel, minus supporting time. The supporting time includes the time of crew mobilization, breaks, dirt removal, lining, resetting, material loading, and dirt dumping. More details can be found in Ourdev, Abourizk, and Al-Bataineh (2007).

This section uses the same project as described in section 4.4 (NEST) for input analysis and updating. After all data is processed, the list of penetration rates is the basis of model inputs. The histogram of the data set is first analyzed, and then the data is fitted to statistical distributions. The best fit distribution will be selected and used as input for the simulation model. One penetration rate data point can be obtained from the progress of a shift. Figure 5-2a shows the analysis of 150 data points. Four distributions are fitted to the data. The Beta distribution and Gamma distribution are almost overlapped; they show a better match to the histogram and are recommended in this study to use as model input. An exponential distribution fitting is also good, but the Normal distribution should not be used.

Model inputs should be continuously updated as more data becomes available. Figure 5-2b shows the input analysis of all 366 data points in this case study project. Beta and Gamma distributions still fit the histogram best. Table 5-1 shows a basic statistical analysis of 150 data points and 366 data points and the results of fitting to Beta and Gamma distributions. The mean penetration rate increased from 0.035 meter per minute to 0.042 meter per minute, which means the TBM production increased 20% from the first 150 data points gathered. The parameters of fitted distribution also changed substantially. However, the standard error of estimated parameters converged as data points increased. For example, in Beta distribution fitting, the *a* parameter changed from 1.051 to 1.312, but the estimation error reduced from 0.139 to 0.100. This demonstrates the value of updating model inputs to maintain valid model inputs, which allows the simulation model to provide reliable long-term support.





a - Input analysis of 150 data points **b - Input analysis of 366 data points**

Figure 5-2 Simulation model input analysis

Table 5-1 Statistical analysis of TBM penetration rate

Distribution Type	**Data Points**	**Mean**	**Variance**	**Parameter *a***	**Std. Error of *a***	**Parameter *b***	**Std. Error of *b***
Gamma	150	0.035	0.001	1.080	0.111	0.032	0.004
	366	0.042	0.001	1.354	0.090	0.031	0.002
Beta	150	0.035	0.001	1.051	0.139	29.360	4.559
	366	0.042	0.001	1.312	0.100	30.298	2.920

5.5 Discussion

Bayesian technique has been applied and developed as a component using COSYE, which is reusable for other HLA-based simulation models. It enables model inputs to be updated dynamically, as more data becomes available. This adaptive modeling approach can improve the reliability of simulation model inputs long-term. The concept of automatically updating simulation model inputs based on the latest project data is not applied in previous construction simulation. Simulation model inputs are usually predefined according to historical data. Previous study shows that only a pilot study of using the conjugate distributions based on Normal distribution was introduced in construction.

This research applied conjugate distributions which uses Gamma distribution as prior. The goodness of fitness has been improved, as discussed in the previous section. As Gamma distribution has a minimum bound, it can guarantee that any sampled data in the simulation is not negative. Negative samples are always a problem when we apply Normal distribution as prior, as in previous studies.

One limitation of this research is that it only applied conjugate distribution in model inputs updating. Although Gamma distribution improved the goodness of fitness of the data sample compared to Normal distribution, the distribution of the sample data could change over time. The function of this input updating module could be enhanced by integrating different distributions. Additionally, this method does not consider the seasonal effect of the data. Applying time series modeling could better forecast model inputs by incorporating the seasonal effects of data, which inherits from construction activities.

5.6 Summary

This chapter describes a Bayesian updating federate that implemented the theory of Bayesian inference. It can automatically update the TBM penetration rate according to the latest collected project data. The purpose is to enhance the quality of simulation model inputs. In this study, the input is TBM penetration rate. The idea of updating simulation model inputs using Bayesian updating technique can be explored in other simulation models.

CHAPTER 6 WEB-BASED DATA COLLECTION FOR AUTOMATING SIMULATION MODEL INPUTS UPDATING

6.1 Introduction

Computer simulation has been widely applied in modeling construction operations to gain insight into project performance. It helps improve project performance, which can result in reduced project cost, improved project quality, and shortened duration (Abourizk 2010). Simulation models are able to provide long-term performance analysis for planning and design, but they are rarely used after the project planning and design stage (Moon and Phatak 2005). One main constraint is the time and effort needed to gather and extract pertinent information for simulation models (Bengtsson et al. 2009; Moon and Phatak 2005; Skoogh et al. 2010). This, to some extent, constrains traditional simulation models from repetitive and long-term usage (Moon and Phatak 2005). The lack of tools and methods for timely communication and interaction between model developers and field engineers is also a limiting factor for simulation models to provide long-term continuous support through project execution.

This research presents a progress monitoring system which was developed to capture progress data, and provide project progress input into a simulation model. The progress monitoring system is a database application for collecting progress data from a tunnel construction site. It feeds the progress data into a tunnel simulation model. The system provides a convenient way to capture site data. This provides a chance to quickly update simulation model inputs by extracting pertinent information according to the latest data. Then the simulation model is able to provide long-term support through overall project execution. In addition to supporting the simulation model, the monitoring system can be used alone for progress analysis. This is discussed in detail in this research. In all, this research demonstrates automatically integrating progress data into a simulation model, and presents a case study for a tunnel construction project.

6.2 Conceptual design

A progress monitoring system is proposed to provide the latest progress data for tunnel federation. The information flow process is described in Figure 6-1. Site engineers can enter data into the progress monitoring system through a personal digital assistant (PDA) or a computer. These data are saved in a database server.

The tunnel simulation system can then retrieve the data and process it to update model inputs.

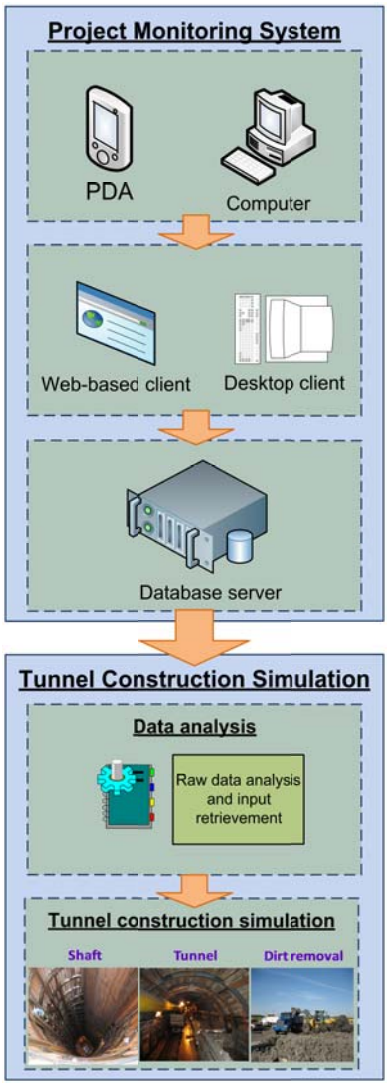

Project Monitoring System
PDA
Computer
Web-based client
Desktop client
Database server
Tunnel Construction Simulation
Data analysis
Raw data analysis and input retrievement
Tunnel construction simulation
Shaft
Tunnel
Dirt removal

Figure 6-1 Conceptual design of distributed tunnel construction monitoring and simulation

6.3 Progress monitoring system

The progress monitoring system is designed to collect and analyze project progress data, and to provide the basis of simulation model inputs. The progress monitoring system is built using the DADE (DRAXware Application Development Engine) framework, a database application development platform which is written in Delphi. It provides two types of database solutions. A web-based solution is developed for users to enter data through a browser. It can be used with both PDAs and computers. A desktop solution also provides data analysis and reporting function to help users understand the ongoing projects.

The web-based client allows users to enter project information through various personal digital assistants (PDAs) such as the Blackberry™ or iPhone™, as long as the Internet is available. This client provides the flexibility to enter data whenever new information is available and captured. It is useful when construction job sites are scattered and information is distributed. The web client focuses on functions of data entry and review.

The desktop client collects the same information as in the web client, but gives a better view of all project information, and allows searching and filtering for data analysis. It should fit project managers' requirements for reviewing and analyzing data, and help them to better understand the project progress.

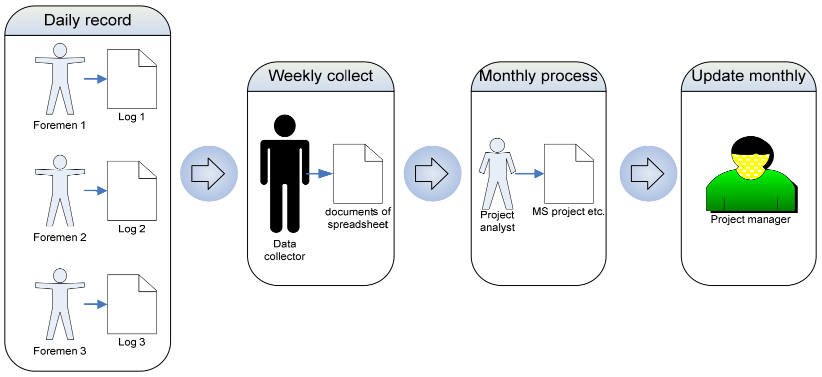
The desktop client provides rich data analysis and report generation in addition to data entry and review as in the web solution. Three types of reports are provided based on daily production, weekly production, and monthly production. The monthly report summarizes the progress of every month for each project. It shows the remaining length of a project, the average production per shift, and the estimated shifts to completion based on the productivity of the current month. In addition, average crew size, average shift length, and total interruption duration of a month are included in the report. Similar to the monthly report, the weekly report and daily report are based on weekly project progress and daily progress, respectively. Graphical analysis is also provided in the progress monitoring system. It provides a better understanding of production changes and production trends. More details such as the interfaces of the system are presented in the case study.

6.4 Project introduction

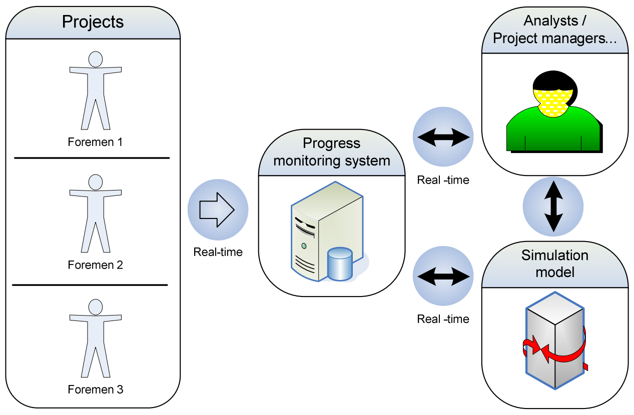
This research is based on the practice of tunnel construction in the City of Edmonton, Alberta by the Design and Construction Section of Drainage Services of the City's own workforce (D&C). This case study project is called the North Edmonton Sanitary Trunk (NEST) NL 2,3, and is located in Edmonton, Alberta, Canada. NEST was selected to represent a typical utility tunnel project; it is 3707 meters in length, and 2.344 meters in diameter. The project started in December 2007 using a tunnel boring machine (TBM). The tunneling operations have been monitored, and data about the production progress as well as all interruptions have been recorded since construction began.

6.5 Process of progress monitoring and data collection

The process monitoring and information is discussed in this section. The information transfer flow in D&C, without using a progress monitoring system, is shown in Figure 6-2a. In this situation, it takes an unacceptably long time to transfer site information to a project manager who is working in the head office, and project managers have limited access to detailed project schedules. It is difficult to track the progression of contract commitments as they occur on a project, which leads to delayed response to cost and scheduling problems and a reactive project management environment. The proposed information flow using the monitoring system is described in Figure 6-2b. The site engineers can record construction information and send it to a database. As a result, the simulation model is able to retrieve the latest progress data. In addition, the project managers and analysts are able to access the latest project progress.



a - The process of traditional information flow

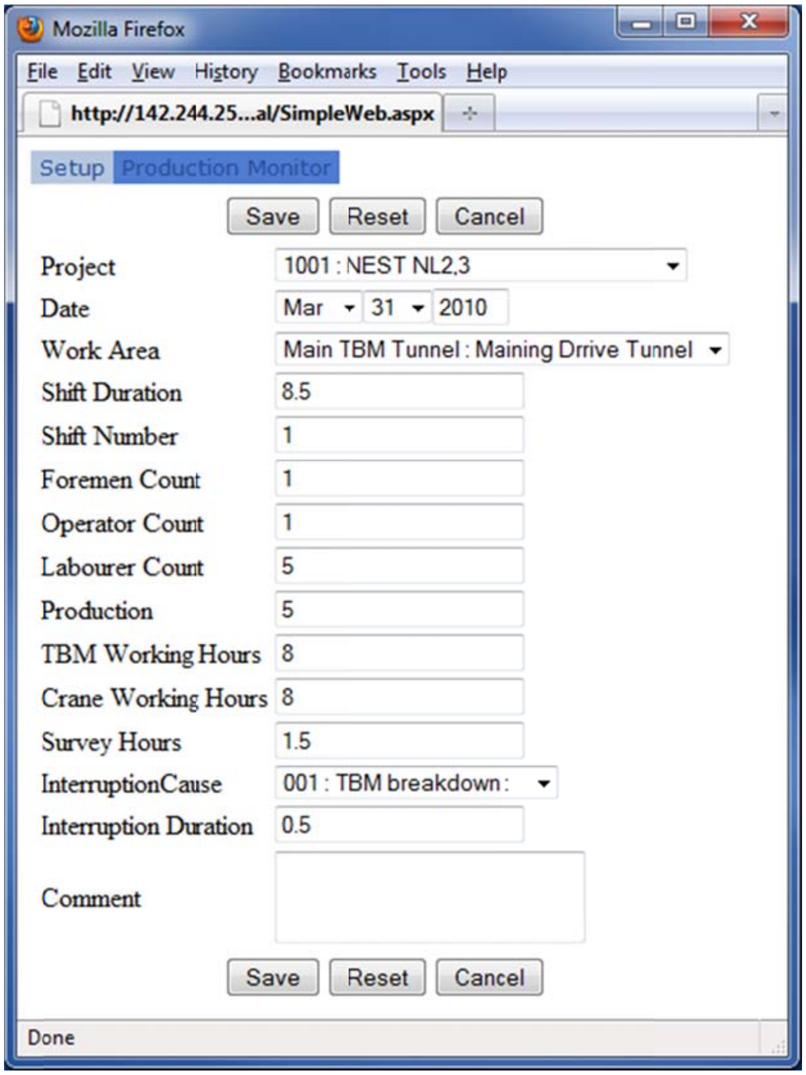


b - The process of proposed information flow

Figure 6-2 Process of data flow

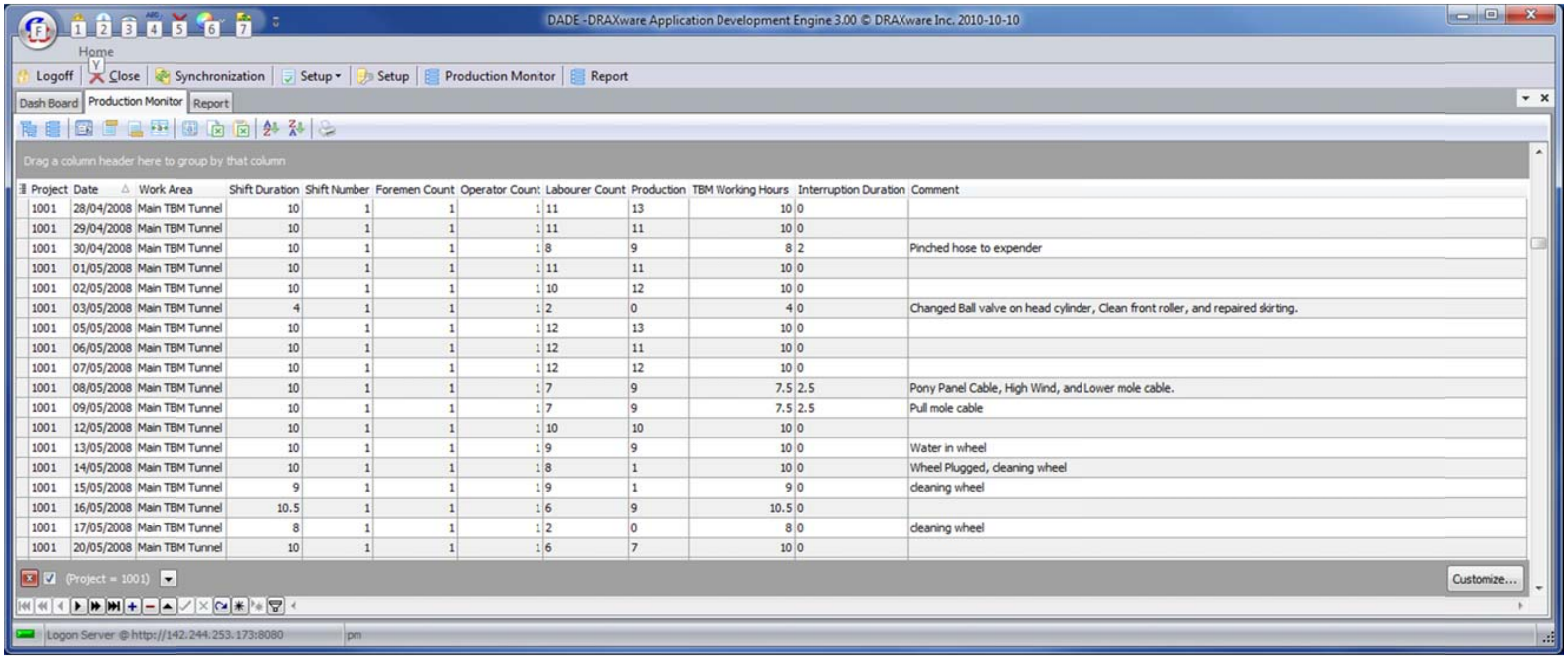
6.6 Data collection and analysis

Both the web-client and the desktop client support progress monitoring and data entry. The data entry interface of the web client is shown in Figure 6-3. Progress data of a tunnel project can be entered here, including shift duration, shift number, number of foremen, number of operators, number of laborers, production, TBM working hours, and so on. The web client focuses on convenient and practical daily usage; it allows inspectors, foremen, and site engineers to simply capture data in a convenient way.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://142.244.25...al/SimpleWeb.aspx
Setup Production Monitor
Save Reset Cancel

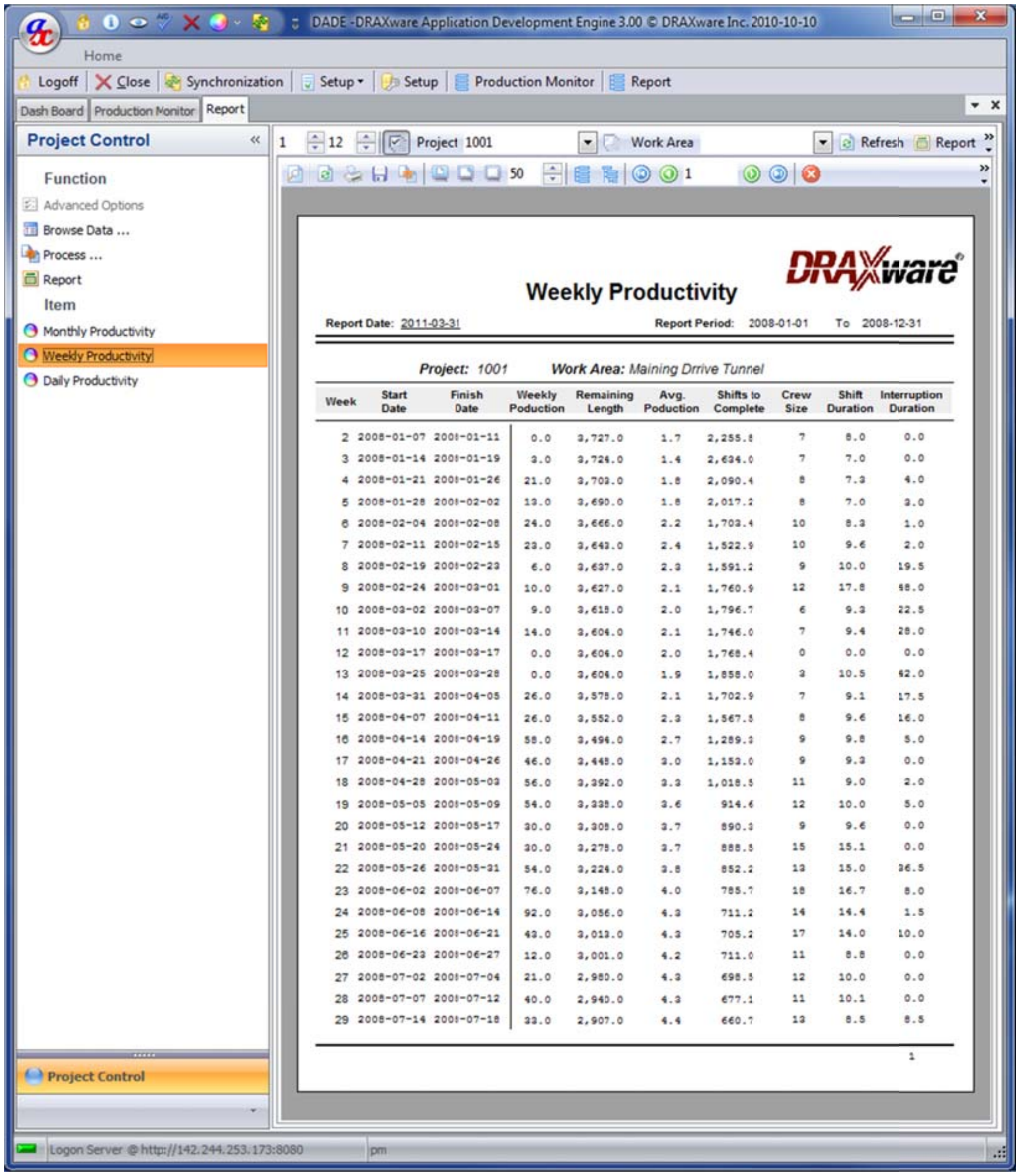
Project	1001 : NEST NL2,3
Date	Mar 31 2010
Work Area	Main TBM Tunnel : Maining Drrive Tunnel
Shift Duration	8.5
Shift Number	1
Foremen Count	1
Operator Count	1
Labourer Count	5
Production	5
TBM Working Hours	8
Crane Working Hours	8
Survey Hours	1.5
InterruptionCause	001 : TBM breakdown :
Interruption Duration	0.5
Comment	

Figure 6-3 The web-based client for progress monitoring


DADE -DRAXware Application Development Engine 3.00 © DRAXware Inc. 2010-10-10
Home
Logoff
Close
Synchronization
Setup
Setup
Production Monitor
Report
Dash Board
Production Monitor
Report
Drag a column header here to group by that column
(Project = 1001)
Customize...
Logon Server @ http://142.244.253.173:8080
pm

Project	Date	Work Area	Shift Duration	Shift Number	Foremen Count	Operator Count	Labourer Count	Production	TBM Working Hours	Interruption Duration	Comment
1001	28/04/2008	Main TBM Tunnel	10	1	1	1	11	13	10	0	
1001	29/04/2008	Main TBM Tunnel	10	1	1	1	11	11	10	0	
1001	30/04/2008	Main TBM Tunnel	10	1	1	1	8	9	8	2	Pinched hose to expender
1001	01/05/2008	Main TBM Tunnel	10	1	1	1	11	11	10	0	
1001	02/05/2008	Main TBM Tunnel	10	1	1	1	10	12	10	0	
1001	03/05/2008	Main TBM Tunnel	4	1	1	1	2	0	4	0	Changed Ball valve on head cylinder, Clean front roller, and repaired skirting.
1001	05/05/2008	Main TBM Tunnel	10	1	1	1	12	13	10	0	
1001	06/05/2008	Main TBM Tunnel	10	1	1	1	12	11	10	0	
1001	07/05/2008	Main TBM Tunnel	10	1	1	1	12	12	10	0	
1001	08/05/2008	Main TBM Tunnel	10	1	1	1	7	9	7.5	2.5	Pony Panel Cable, High Wind, and Lower mole cable.
1001	09/05/2008	Main TBM Tunnel	10	1	1	1	7	9	7.5	2.5	Pull mole cable
1001	12/05/2008	Main TBM Tunnel	10	1	1	1	10	10	10	0	
1001	13/05/2008	Main TBM Tunnel	10	1	1	1	9	9	10	0	Water in wheel
1001	14/05/2008	Main TBM Tunnel	10	1	1	1	8	1	10	0	Wheel Plugged, cleaning wheel
1001	15/05/2008	Main TBM Tunnel	9	1	1	1	9	1	9	0	cleaning wheel
1001	16/05/2008	Main TBM Tunnel	10.5	1	1	1	6	9	10.5	0	
1001	17/05/2008	Main TBM Tunnel	8	1	1	1	2	0	8	0	cleaning wheel
1001	20/05/2008	Main TBM Tunnel	10	1	1	1	6	7	10	0	

Figure 6-4 The desktop client for progress monitoring



Weekly Productivity

Report Date: 2011-03-31   Report Period: 2008-01-01 To 2008-12-31

Project: 1001   *Work Area:* Maining Drrive Tunnel

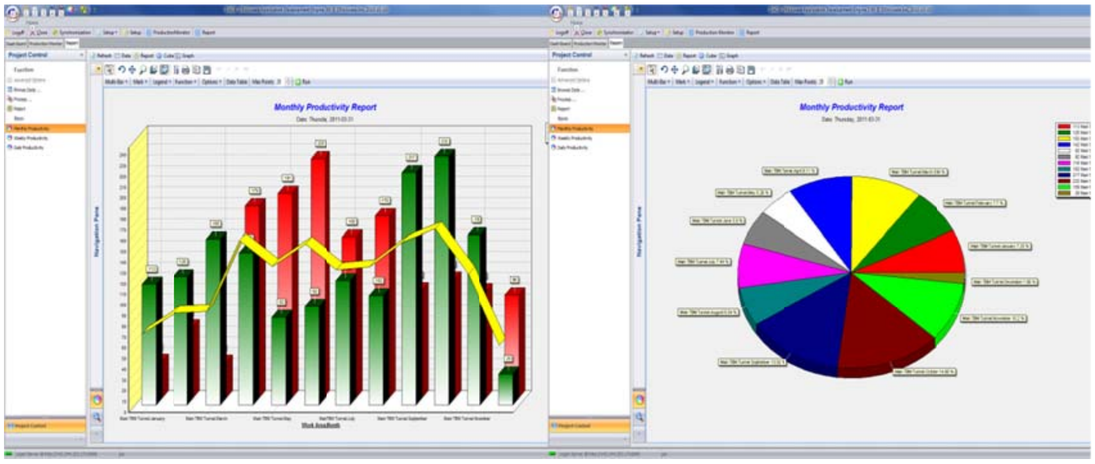
Week	Start Date	Finish Date	Weekly Poduction	Remaining Length	Avg. Poduction	Shifts to Complete	Crew Size	Shift Duration	Interruption Duration
2	2008-01-07	2008-01-11	0.0	3,727.0	1.7	2,255.8	7	8.0	0.0
3	2008-01-14	2008-01-19	3.0	3,724.0	1.4	2,634.0	7	7.0	0.0
4	2008-01-21	2008-01-26	21.0	3,703.0	1.8	2,090.4	8	7.3	4.0
5	2008-01-28	2008-02-02	13.0	3,690.0	1.8	2,017.2	8	7.0	3.0
6	2008-02-04	2008-02-08	24.0	3,666.0	2.2	1,703.4	10	8.3	1.0
7	2008-02-11	2008-02-15	23.0	3,643.0	2.4	1,522.9	10	9.6	2.0
8	2008-02-19	2008-02-23	6.0	3,637.0	2.3	1,591.2	9	10.0	19.5
9	2008-02-24	2008-03-01	10.0	3,627.0	2.1	1,760.9	12	17.8	48.0
10	2008-03-02	2008-03-07	9.0	3,618.0	2.0	1,796.7	6	9.3	22.5
11	2008-03-10	2008-03-14	14.0	3,604.0	2.1	1,746.0	7	9.4	28.0
12	2008-03-17	2008-03-17	0.0	3,606.0	2.0	1,768.4	0	0.0	0.0
13	2008-03-25	2008-03-28	0.0	3,604.0	1.9	1,858.0	3	10.5	42.0
14	2008-03-31	2008-04-05	26.0	3,578.0	2.1	1,702.9	7	9.1	17.5
15	2008-04-07	2008-04-11	26.0	3,552.0	2.3	1,567.5	8	9.6	16.0
16	2008-04-14	2008-04-19	58.0	3,494.0	2.7	1,289.3	9	9.8	5.0
17	2008-04-21	2008-04-26	46.0	3,448.0	3.0	1,153.0	9	9.3	0.0
18	2008-04-28	2008-05-03	56.0	3,392.0	3.3	1,018.5	11	9.0	2.0
19	2008-05-05	2008-05-09	54.0	3,338.0	3.6	914.6	12	10.0	5.0
20	2008-05-12	2008-05-17	30.0	3,308.0	3.7	890.3	9	9.6	0.0
21	2008-05-20	2008-05-24	30.0	3,278.0	3.7	888.5	15	15.1	0.0
22	2008-05-26	2008-05-31	54.0	3,224.0	3.8	852.2	13	15.0	36.5
23	2008-06-02	2008-06-07	76.0	3,148.0	4.0	785.7	18	16.7	8.0
24	2008-06-08	2008-06-14	92.0	3,056.0	4.3	711.2	14	14.4	1.5
25	2008-06-16	2008-06-21	43.0	3,013.0	4.3	705.2	17	14.0	10.0
26	2008-06-23	2008-06-27	12.0	3,001.0	4.2	711.0	11	8.8	0.0
27	2008-07-02	2008-07-04	21.0	2,980.0	4.3	698.5	12	10.0	0.0
28	2008-07-07	2008-07-12	40.0	2,940.0	4.3	677.1	11	10.1	0.0
29	2008-07-14	2008-07-18	33.0	2,907.0	4.4	660.7	13	8.5	8.5

1

Figure 6-5 Weekly progress report in progress monitoring system

The desktop client provides more functions of data analysis and reporting. The interface for data entry is shown in Figure 6-6. An example of a weekly report is presented in Figure 6-7. Figure 6-8a shows the monthly production from 2008 to

2009 in a bar graph format. An average production line is added for trend analysis. Figure 6-8b shows the production of 2009 in a pie chart.



a - Monthly production bar graph **b - Monthly production pie chart**

Figure 6-6 Examples of graphic analysis in progress monitoring

6.7 Conclusion

The main purpose of the study is to demonstrate the integration of progress input data into a distributed tunnel construction simulation model. This research presented a progress monitoring system for monitoring and analyzing tunnel progress. Information from scattered construction sites is captured and recorded in the database through the system. The web client allows a user to capture data using PDAs such as the iPhone™. The reporting and analysis function from the desktop client is also presented in this paper. The progress monitoring system works in conjunction with an HLA-based tunnel construction simulation, contributing to a tunnel simulation model with the latest available progress data. Thus, the tunnel simulation model is able to provide long-term reliable decision support. The data collection system has been applied to a tunnel construction project to illustrate how the data monitoring system can ease the time and effort needed in capturing scattered data and retrieving model inputs.

CHAPTER 7 ENHANCING QUALITY OF COST CONTINGENCY FORECAST USING VAR METHOD

7.1 Introduction

Cost overrun is a significant challenge in construction projects. A statistical study on a sample of 258 transportation infrastructure projects worth US$90 billion shows that the actual costs are on average 28% higher than estimated for all types of projects, 34% higher for all fix-link projects (tunnels and bridges), 45% higher for all rail projects and 20% higher for all road projects (Flyvbjerg et al. 2002). Construction megaprojects show spectacular examples of cost overrun. For example, the Sydney Opera House cost 15 times its budget (Flyvbjerg et al. 2002). Flybjerg (2007) noted that Boston's "Big Dig" central artery/tunnel construction project was 275% ($11 billion) over budget, and the Channel Tunnel between the UK and France had an 80% construction cost overrun.

This research proposes a simple but practical method based on Value at Risk (VaR) to determine and update project contingencies during project execution. As suggested by Park et al. (2005), a good forecasting technique accounts for both a historical performance and competent judgments based on construction experience and knowledge. The VaR technique allows project managers to incorporate their experience by specifying an appropriate period of historical data and a period of future performance in the analysis.

The method is demonstrated through a case study of a tunnel construction project using as-built data of daily progress and actual daily cost from accounting records of the project; it is also applicable to projects where the progress is measured on other timescales, such as weekly or monthly. Predictions are generated, updated, and reissued at project milestones, according to the actual performance of a project when new information is available.

7.2 Background

7.2.1 Value at Risk definition

Value at Risk (VaR) is a statistical measure of the maximum loss that can be expected to occur with a given probability, over a specified period of time, under normal conditions. Probability is specified in terms of the confidence level *c*. For example, a bi-weekly VaR of \$7,000 at a 90% confidence level means that in any two-week period there is a 10% (100%–90%) chance that the losses will equal or exceed \$7,000. Alternatively, we could say that we should expect in a two-week period to have one workday (1/10th chance) on which the losses will be at least \$7,000. VaR has been widely used since the mid-1990s (Damodaran 2008), and a detailed introduction can be found in Jorion (1997).

This research implements an analytical or variance/covariance method of VaR. The method assumes that the historical data are a sample from a probability distribution such as a Normal distribution. The confidence interval of 80%, 90%, and 98% in a Normal distribution refers to the areas of $\pm 1.2816\sigma$, $\pm 1.6449\sigma$ and $\pm 2.3263\sigma$ under the distribution curve. Because VaR is concerned with negative risk or loss, not with positive risk or gain, we ignore the positive side. Thus, $\pm 1.2816\sigma$, $\pm 1.6449\sigma$, and $\pm 2.3263\sigma$ correspond to the confidence levels of 90%, 95%, and 99%, respectively, in VaR. This interpretation directly leads to the definition of the parametric value at risk:

$$VaR_c = Z_c\sigma \qquad (7\text{-}1)$$

Where,

VaR_c denotes the value at risk at confidence level of c;

c denotes the confidence level, such as 95%, $c \in (0,1)$;

$Z_c$denotes the critical value corresponding to confidence level c, the inverse of the standard normal cumulative distribution with cumulative probability of 1-c, i.e. $Z_{0.95} = -1.645$;

σ' denotes the volatility as a percentage of the whole asset value;

Wdenotes the value of the whole asset;

σ denotes the volatility of the whole asset in dollar value.

This yields the dollar loss in the value of an asset that would be exceeded only 1-c percent of the time. The result from Equation (7-1) is also called relative VaR, because it describes the value below the expected return. Another definition of VaR is absolute VaR, which is calculated by deducting the expected return from relative VaR, as in Equation (7-2).

$$VaR'_c = Z_c\sigma - \mu \tag{7-2}$$

Where μ is the expected return.

The results calculated using Equation (7-1) represent the value at risk for a time horizon of the next period, usually daily. The assumption of the Normal distribution also provides a formula for easy conversion between VaR of different time horizons, shown in Equation (7-3).

$$VaR_{c,T} = VaR_{c,1} * \sqrt{T} \tag{7-3}$$

Where,

$VaR_{c,1}$ denotes the value at risk for time horizon of next 1 period, equivalent to VaR_c in Equation (7-1);

T denotes the number of time periods;

$VaR_{c,T}$ denotes the value at risk for time horizon of next T period.

7.2.2 VaR properties

VaR accounts for loss probability, time horizon, and historical performance, and is widely used in financial risk management. A risk manager can use it to define risk threshold for risk control and risk management purposes. This research applied VaR to predict the probabilistic potential loss and update the project contingencies. Both project cost and project progress were analyzed for assessment and prediction. According to the Project Management Institute (2004), predictions are typically generated, updated, and reissued based on the past performance of a project and any new information available when a project is executed and progresses. VaR also provides a chance to make use of newly available data as a project progresses and update forecasts based on that data.

There are three factors used in calculating VaR: the prediction period, the confidence level *c*, and the observation period or the size of the sample. The prediction period is referred to as the "time horizon" in financial risk management. In this research it reflects the time frame for the projection of performance and the desired control period for project managers. A time horizon may range from a single workday to a successive period of ten workdays, all the way up to one calendar year. The preferred time horizon often depends on the type of project and the management style. The confidence level *c* is the probability that the actual value will reach or exceed the predicted maximum loss. It represents organizations' tolerance level for risks. For example, in finance, the Banker Trust uses a 99% confidence level, J.P. Morgan uses 95%, and Citibank uses 95.4% (Dupačová et al. 2002). The third parameter for calculating VaR is the observation period. In this research it is the data sample size. Understanding project performance and its possible changes is a prerequisite for deciding sample size and projecting performance. Because the VaR model assumes a Normal distribution of the sample, the sample size must be large enough to satisfy this assumption, depending on the variability of the data and the desired margin of error.

All these three factors affect the accuracy of VaR, which provides a chance for the incorporation of project managers' experience, knowledge, and management style. Performance information which is too old may not have much correlation with current performance; defining an appropriate period of past performance as a sample for VaR relies on the experience and judgment of project managers according to the project type and its properties. Too small a sample may not be representative, but too large a sample may include irrelevant information and decrease accuracy. In our case study, for example, as the project progresses, data for the period of 100 workdays immediately before the prediction date is used as input, and the data older than that is ignored. The user can easily change this to any other time period, according to project type and properties. Our process implements the principle that the latest performance has a higher impact on future performance than out-dated performance.

7.2.3 VaR inputs and outputs

Project profit/loss is different from cash flow and cost. These concepts are sometimes not clearly defined in project management. Project cost refers to various costs needed to complete a project. In finance, cash flow shows the money flowing into a business from sales, interest payments received, and any borrowing, and the amount of money flowing out of a business for wages, rent, interest owing, paying back loans, buying raw materials and so on. If the cash flowing into a business does not meet the cash flowing out, then a company could be forced out of business because it is not able to serve its debts. Cash flow is greatly affected by payment type and payment time delay. In engineering, revenue is usually received with a time lag after certain progress goals are achieved and approved, but potential earning and cost occur incrementally every day. Daily profit is therefore estimated based on project progress.

Project daily profits/losses are analyzed and used as inputs in VaR models in this research. The project daily profits/losses are calculated from project daily progress and cost. For example, consider a small manufacturing factory. The factory sells products at $100 each with a marginal profit rate of 20%, and is able to manufacture 40 to 50 products every day using a machine. If the key machine breaks down, and the factory cannot manufacture any product, there is still some overhead and other indirect costs like machine repair. The cost contingency is used to cover these overhead and indirect costs. If the breakdown happened after 10 products were made, the potential loss of that day will be the actual cost minus the income made from the 10 products. In this situation, what is the contingency used? Note that the cost contingency is defined as the cost that is used to cover unforeseen work due to risks, in addition to estimated project cost. Therefore, the profit should be excluded. If the cost of that day is -3000 (i.e., 3000 expense), then the contingency is -2,200 (-3000 – (-100 · 10 · (1 – 20%)) = -2,200). This daily amount is used as input in the VaR model, and called "project daily profit/loss" in this research. It is defined as the estimated potential earning of daily work performed (excluding estimated profit) minus actual daily cost of work performed. This is a potential profit/loss (loss when negative), since the profit is not actually achieved until payment occurs. Even though the contingency should be negative to indicate a cost overrun, there is sometimes no contingency used, so the "contingency" turns out to be a positive number. Both positive and negative numbers are used as input in our calculation.

A previous study forecasted cost contingency as both a dollar value and a percentage of the cost estimate using an artificial neural network (ANN) (Lhee et al. 2009). The ANN model was trained, tested, and validated using real data from 495 projects and showed that the contingency amount forecast had a higher correlation, fewer errors, and better results than a percentage forecast (Lhee et al. 2009). Therefore, in this research, the VaR model forecasts the contingency amount.

The results of VaR models are directly considered as cost contingencies. According to the VaR definition, it forecasts the maximum loss at a probability level over a period. The loss occurs due to various risks and needs to be covered by the cost contingency. This research analyzes project daily loss, considered as the impacts of unforeseen risks, instead of risks themselves, then forecasts the likely loss for future and the contingency required. The contingencies are presented for a coming period, such as half a year or a year, depending on project duration and the company plan.

7.3 VaR modeling and contingency updating

VaR modeling and contingency updating consist of a circular calculation, implemented in this research via a simple spreadsheet. The process is presented in Figure 7-1. All tasks with a filled background are parts of VaR modeling, and the other tasks are supportive. The tasks in the dashed rectangular box should be repeated at the end of every observation period to forecast contingency. The following will explain the process of calculating contingency step by step:

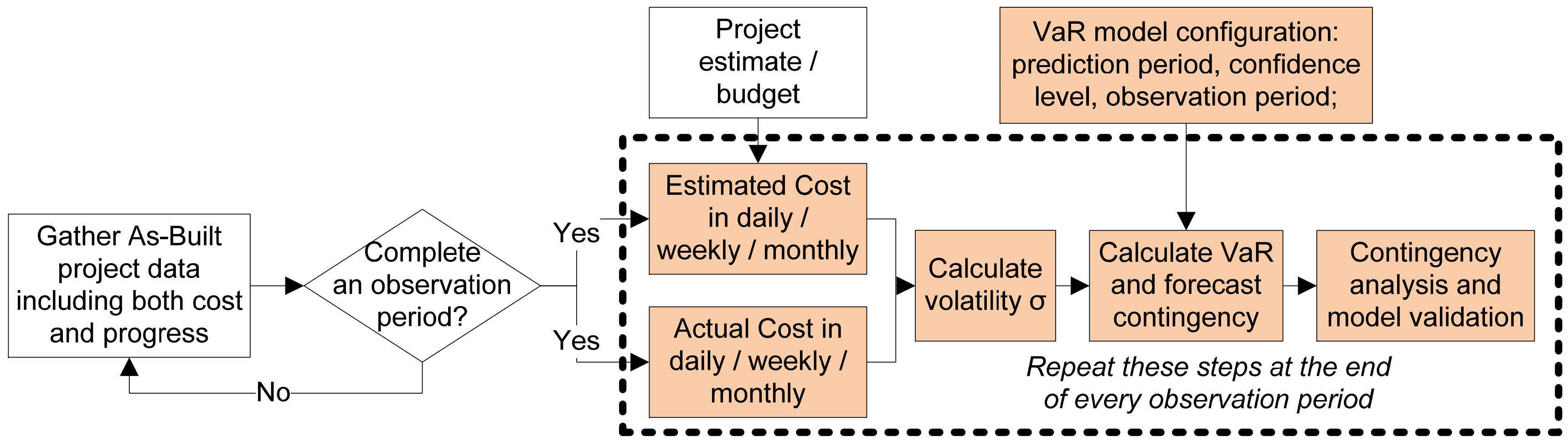

Project estimate / budget
VaR model configuration: prediction period, confidence level, observation period;
Gather As-Built project data including both cost and progress
Complete an observation period?
Yes
Yes
No
Estimated Cost in daily / weekly / monthly
Actual Cost in daily / weekly / monthly
Calculate volatility σ
Calculate VaR and forecast contingency
Contingency analysis and model validation
Repeat these steps at the end of every observation period

Figure 7-1 VaR modeling process

Gather project information and define VaR parameters. At the beginning of a project, project estimate, budget, and estimated profit should be gathered and are used as basis in calculating volatility and contingency. Meanwhile, contingency prediction period, confidence level, and observation period are decided, according to project properties, uncertainty levels, and so on.

Once a project commences, as-built project data should be continuously gathered, including both project progress and actual expenses. Companies usually have their own systems to record cost for financial management purposes. These records provide the fundamental basis for VaR analysis.

When completing an observation period, we analyze the data and reorganize it in daily, weekly, monthly, or other timescales in project profit/loss analysis. Project daily potential profit/loss is calculated by deducting actual daily cost of work performed from the estimated earning of daily work performed (excluding estimated profit), and is used as VaR model input. Actual cost analysis includes, but is not limited to, removing "dummy" entries in bookkeeping, summing up expenses, and apportioning occasional large expenses to appropriate period. In order to calculate estimated earning of work performed, the work performed is first quantified by measuring project progress. Then the estimated earning of work performed can be obtained from actual progress and project estimate. The project progress, actual cost and estimated cost can be measured in a consistent timescale, such as daily, weekly, or monthly. Examples of data analysis can be found in the case study.

Then we can calculate volatility. The volatility, σ is first calculated using Equation (7-4), if we use daily data from the past k workdays as inputs:

$$\sigma = \sqrt[2]{\frac{1}{k-1}\sum_{i=-k}^{-1}(profit_i - \mu)^2} \tag{7-4}$$

where $profit_i$ means the project daily profit/loss for day *i,* ranging from -k to -1, and μ denotes the average of $profit_i$ during observation period in dollar value. For example, $profit_{-10}$ represents the daily profit/loss made at the 10th day prior to today. The input size can be easily changed according to project properties and how much information is available.

Value at Risk is calculated based on Equation (7-1) and (7-3). Then we can calculate VaR using Equation (7-1), and then convert it to different time period using Equation (7-3). Using this conversion, we can forecast the contingency for a long period. An example of using daily cost is demonstrated in the case study of this paper. The forecasted contingency can be evaluated by comparison with the actual contingency spent. More details about contingency analysis and evaluation are presented in the sections of VaR calculation, results analysis, and model validation under case study.

7.4 Case study

7.4.1 Project introduction

The chosen project is a section of a utility tunnel construction project, the North Edmonton Sanitary Tunnel, called NEST NL 2,3. It is being built in Edmonton, Alberta, Canada, using a tunnel boring machine (TBM). The project involves excavation of 3.675km of sanitary sewer tunnels, starting in December 2007. The production data for the test case is the information collected continuously from the operation site of the NEST NL 2,3 tunnel project on a daily basis. Other information like shift length, crew size, and temperature, is also recorded in detail on a daily basis. The information constantly monitored from the daily operations provides the basis for the estimation of short term delay in the research.

As VaR reflects the maximum loss, the case study focuses on applying VaR to analyze the potential profit/loss for NEST NL 2,3, based on the contractor's historical project performance. A contractor usually conducts a project according to their bidding documents. The bid price is usually the dollar value that the contractor will ultimately receive from project sponsor(s) for a project if the project is accomplished as in their agreements. The payment from a client (i.e., the revenue for a contractor) is provided according to progress. In other words, the quantified work done represents contractor revenue or contractor earnings. For example, in a construction project of a sanitary tunnel, the length completed per day represents a dollar value that is achieved and earned on that day.

In the research, the analysis accounts for various expenses both directly and indirectly related to NEST as project cost on a daily basis. The bid price defines the baseline of payment that the contractor will get from the client. The unit price of NEST, estimated as cost per meter in the bid, is used for calculating revenue. Thus, the revenue for one day here is quantified as unit price times meters accomplished on that day. The following illustrates how we analyze the data and estimate future loss as a cost contingency.

7.4.2 Data collection

NEST started construction on December 3, 2007. We collected both cost and productivity data from the first day of construction until January 2010; 547 work days in total. The first step was to gather the data necessary for the statistical analysis and development of a VaR model. The cost data are all from SAP enterprise software currently used by the City of Edmonton. Every cost item occurring on a construction site is recorded in SAP, with details of date, activity name, activity duration, cost rate, conducted employee name and so on. We collected more than 40,000 records from the contractor dated December 3, 2007 to January 2010. The contractor recorded an average of 70 cost items each work day. Examples of excerpted cost records used in the research are shown in Table 7-1 and include material cost data (purchase orders), labour cost (per worker per day, along with their work duration, cost code, etc.), equipment cost (purchase, rental, maintenance, fuel consumption, etc.), engineering cost (workshops and consulting) and general overhead cost. Other information, including project estimate information and production information (between 3 to 8 meters per shift), is collected as well. The total cost per meter was estimated at $3,104, which includes crew, equipment, material, subcontractor, and other costs.

Table 7-1 Examples of excerpted cost records used in the research

Object	Cost Element	Cost Element Name	Cost	Document Date
130659	655110	Engineering Fees	537.08	03/03/2008
130659	663910	Project Officer	195.3	03/03/2008

133415	420230	Hired Equipment	960	03/03/2008
133415	420230	Hired Equipment	247.5	03/03/2008
133415	663176	Welder I	46.42	03/03/2008
133415	663176	Welder I	92.84	03/03/2008
133415	663182	Welder Foreman	53.01	03/03/2008
133415	663182	Welder Foreman	73.12	03/03/2008
133415	663198	Equip Operator III	285.92	03/03/2008
133415	663199	Equip Operator III OT	73.94	03/03/2008
133415	663200	Equip Operator IV	303.28	03/03/2008
133415	663200	Equip Operator IV	78.45	03/03/2008
133415	663248	Tunnel Labourer I	267.12	03/03/2008
133415	663248	Tunnel Labourer I	69.09	03/03/2008
133415	663248	Tunnel Labourer I	-267.12	03/03/2008
133415	663248	Tunnel Labourer I	267.12	03/03/2008

7.4.3 Data analysis

The data was then cleaned and reorganized on a daily basis; accounting, used in this case project, records the actual cash flow, not the incremental daily costs. For example, materials are paid for in a lump sum, but consumed over a period of months. All records are thus aggregated based on the date, and represented as the cost of every day. The summed-up cost along with productivity on a daily basis is illustrated in an example shown in Table 7-2. In addition, large occasional purchases, such as equipment purchased on 16/01/2008 at a cost of $75,185 (Table 7-2); "dummy" entries, which are used for error correction in bookkeeping; and some incidental weekend work-related costs were identified and apportioned equally to every workday, in order to reflect the real daily cost. Cost items excluded from daily cost which need apportioning include concrete segment cost, engineering consulting, construction consulting, power charges, general-purpose vehicle leases, internal order overhead, general contract work, equipment purchase, power charges, and all other costs that occurs at any non-workday. In our case study project, these costs were summed every three months, and the total was equally distributed to every workday of those three months. For example, these expenses during June 1 2008 to 31 August 2008 sum up to $277,431, and there are 70 workdays during this period. The average per workday becomes $3,963. All these cost items were first deducted from their original bookkeeping dates, and then the average cost, $3,963 is added to every workday cost after deduction. Thus, the total cost is consistent with the original bookkeeping. In order to maintain some level of uncertainty in the research, occasional expenditures like small tool purchases were kept as is to absorb and reflect project risks.

Table 7-2 An example of daily cost and production information

Date	Production	Daily revenue	Actual cost
11/1/2008	0	$ -	$ 5,620
14/01/2008	0	$ -	$ 5,685
15/01/2008	0	$ -	$ 7,966
16/01/2008	0	$ -	$ 75,185
17/01/2008	1	$ 3,104	$ 6,307
18/01/2008	2	$ 6,209	$ 6,108
19/01/2008	0	$ -	$ 5,601

7.4.4 Model inputs

To analyze project daily potential profit/loss, the estimated earning of daily work performed (excluding estimated profit) is first calculated. It is equal to the tunnel daily progress (measured in meters per day) multiplied by the estimated tunnel unit cost (in dollars per meter). Tunnel project progress is recorded by foremen from the tunnel site on a daily basis, and reflects the value of accomplished work immediately after it is produced. Both project cost and earnings are measured in dollars and used as input in contingency updating.

Actual cost takes into account both direct and indirect costs related to the project, such as crew cost, equipment cost, material cost, and indirect cost. The project potential earning means the equivalent dollar value of the progress made in a period, even though the amount of money has not been paid. For example, in our case study, the progress is measured as daily excavated meters of the tunnel, multiplied by a unit cost as estimated before actual construction.

Figures 7-2 and 7-3 show the daily cost, cumulative cost, cash flow and daily potential profit/loss of our case study, the NEST tunnel. The daily potential profit and daily cost in Figure 7-2 are almost overlapped with the x-axis, because they are marginal compared to cumulative cost and cash flow. Figure 7-3 only depicts daily potential profit/loss and daily cost for a closer view.

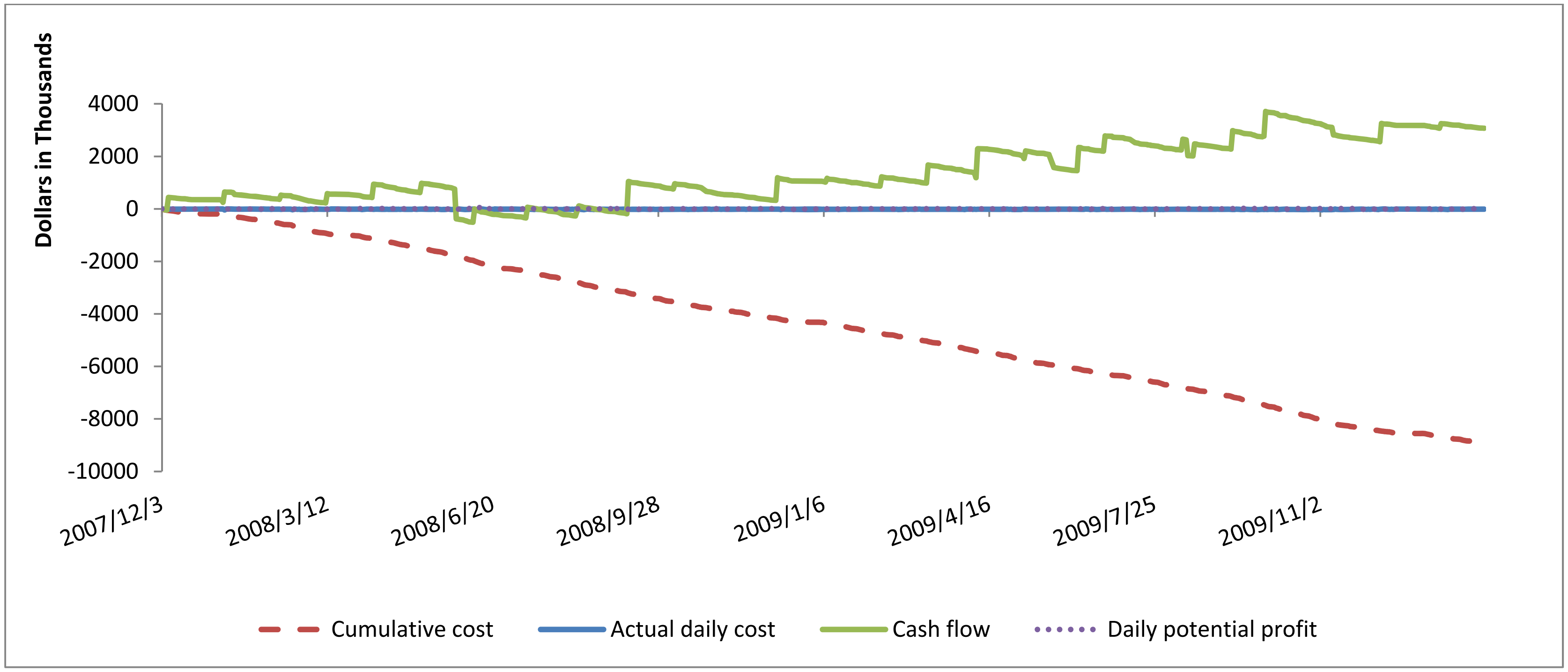


Figure 7-2 Cost related analysis of NEST

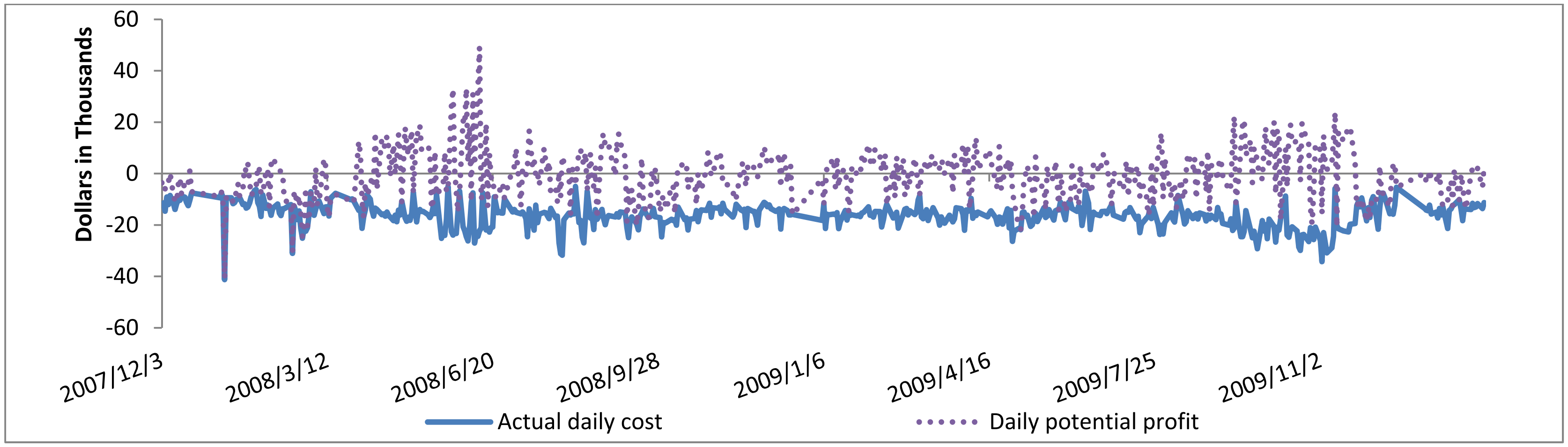


Figure 7-3 Daily cost and daily potential profit/loss of NEST

The cumulative cost increases linearly over the long term, while cash flow gradually increases. Although both cumulative cost and cash flow show NEST performs very well with steady cost and profit, they do not provide details of near future performance to help project managers with short term resource planning and contingency preparation. Again, this research focuses on the potential profit/loss on a daily basis, so as to indicate project performance in a timely manner.

7.4.5 VaR calculation

This section contains a demonstration of how relative VaR was calculated in this case project. For example, suppose we estimate VaR on September 9, 2008 using 200 inputs, which means an observation period of 200 workdays. The profit/loss of the last 200 work days before September 9, 2008, which covers work days from December 6, 2007 to September 8, 2008 according to site records, are used as input data. The volatility is calculated as the standard deviation of these input data using Equation (7-4), which is 13,460 during this observation period. The VaR of September 9, 2008, with *c* of 0.95 (if *c* is set to 0.95, then $Z_{0.95}$ is -1.645) is calculated as follows:

$$VaR_{c,1} = Z_c\sigma = -1.645 * 13{,}460 = -22{,}140$$

The relative VaR of next 200 days is calculated as

$$VaR_{c,T} = VaR_{c,1} * \sqrt{T} = -22{,}140 * \sqrt{200} = -313{,}101$$

The "last 200 work days" is a moving frame of reference, so if the estimate is made one day later, on September 10, 2008, data from December 7, 2007 to September 9, 2008 are used as samples.

7.4.6 Results analysis

Cost contingency can be updated using different sample sizes of historical data at different confidence levels. If we consider every 100 workdays as a project milestone, we can update VaR at each milestone. All or a partial set of the historical data can be used to forecast. For example, if we are at day 301, and 300 data samples of past performance have been collected, then we can use all 300 samples as inputs, or only the last 100 data samples. The key is to use historical data related to current and future performance. If there is a reason to believe that future performance will be more similar to the performance of the past 100 workdays, rather than all 300 workdays, then only the last 100 data samples will be used.

This research used the data from the first half year to forecast the next half year and the next year, for illustration. As tunnel productivity is greatly affected by weather in cold weather areas like Edmonton (Shahin 2007), it is reasonable to believe that productivity has seasonality. A random choice of input size, e.g., 100 or 200, might disturb the seasonality pattern. Thus, annual or semi-annual data was selected as input in forecasting. The project started on December 3, 2007, so we update contingency for a year or half a year later, using all data records through that year or half a year. The first half a year has records of 129 days, and the first year has records of 262 days, including all usual workdays and some overtime during weekends. We can update project contingency at any time necessary. It is not restricted to a specified period after the project starts. The input size can also be easily changed, as long as the selected period is considered to appropriately image the forecasted period.

The results are shown in Table 7-3 with a confidence level of 0.95, as selected in the paper; that is, those levels of contingency would be exceeded only 5% of the time. If updating the contingency for 100 consecutive workdays, we can get a set of 100 forecasted data points and a set of 100 actual samples. Here we have 547 total data samples, and try three different combinations of input data and forecast period.

Table 7-3 Cost contingency using annual or semi-annual data

Estimate made by date	Input size	Estimate Period		
		Next half a year	Next year	Next 1.5 years
6/3/2008	129 (half a year)	-214,478	-305,660	-368,596
12/3/2008	262 (one year)	-209,024	-297,888	-359,223

Case 1: Use 129 data samples to forecast the next 129 workdays, so there are 289 forecasted data points (547-129-129=289) which can be verified;

Case 2: use 129 data samples to forecast the next 262 workdays, so there are 156 forecasted data points (547-129-262 =156) which can be verified;

Case 3: use 262 data samples to forecast the next 129 workdays, so there are 156 forecasted data points (547-262-129=156) which can be verified.

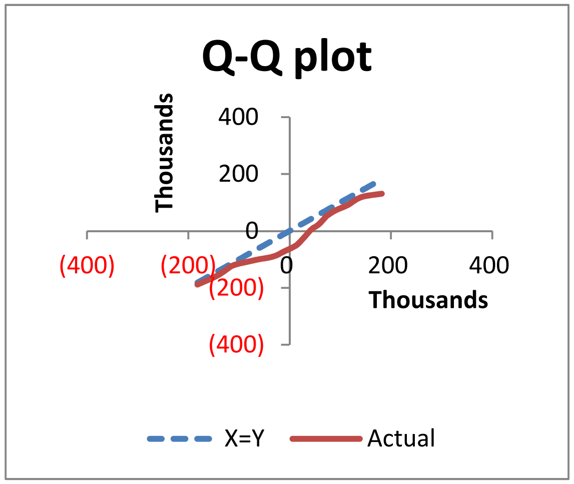
In Case 1, we got 289 forecasted data points with corresponding sample data for comparison. Table 7-4 shows the actual and the forecasted contingencies (α = 0.95) from July 31, 2009 to August 8, 2009 for a period of 129 workdays. In Table 4-4 the forecasted contingencies are all lower than the actual; this is to be expected, because the actual contingency will not exceed the forecast at 95% probability. Similar results were obtained for Cases 2 and 3.

Table 7-4 Contingency ($\mathrm{VaR}_{0.95,129}$) from July 31, 2009 to August 8, 2009

Forecast Date	Actual contingency of next 129 workdays	Forecasted contingency
7/31/2009	66,782	-146,372
8/4/2009	60,898	-147,057
8/5/2009	70,306	-146,663
8/6/2009	74,071	-146,592
8/7/2009	78,858	-146,601
8/8/2009	92,435	-146,635

7.4.7 Model validation

Both Q-Q plots and cumulative distribution functions of actual and forecasted contingencies were used to compare the actual contingency used for the project with the forecasted contingencies (Figures 7-4, 7-5, and 7-6). In a Q-Q plot, the line of $x=y$ is used as reference line. The other line, usually curved, shows a comparison of the forecasted and the actual. If the curved line is below the reference line, the forecast is lower than the actual, and vice versa. All plots show a good visual fit between actual and forecast. The best matching case is the 1st, forecasting for 129 data points using 129 data samples. The 3rd case, forecasting the same period as the 1st case but using more input data, does not show closer matching or better forecasts.



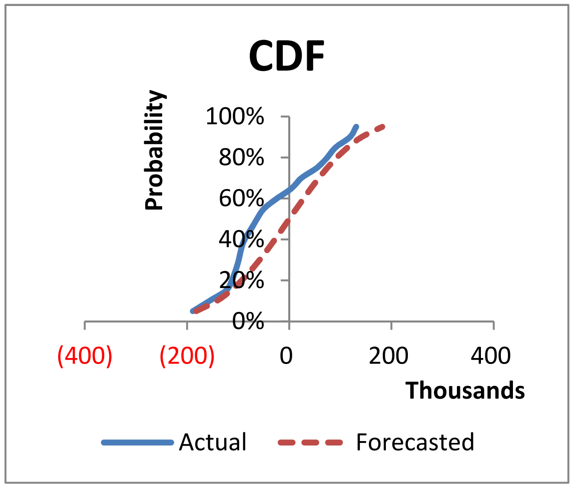
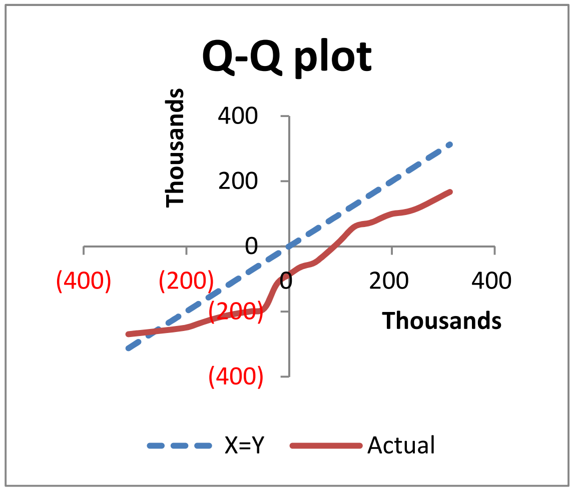


Figure 7-4 Actual contingency and forecasted contingency of Case 1 (input size: 129; forecast period: 129)



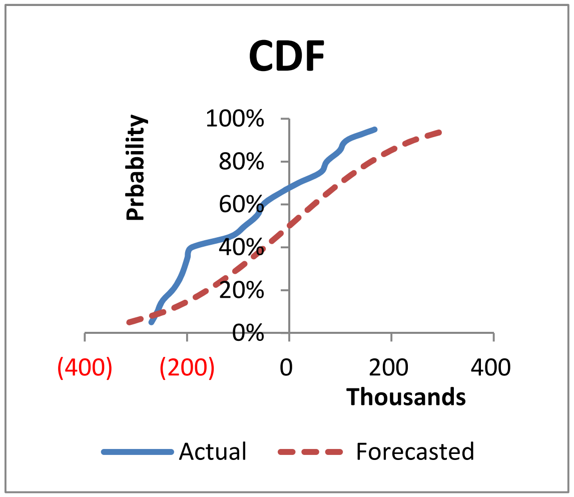


Figure 7-5 Actual contingency and forecasted contingency of Case 2 (input size: 129; forecast period: 262)

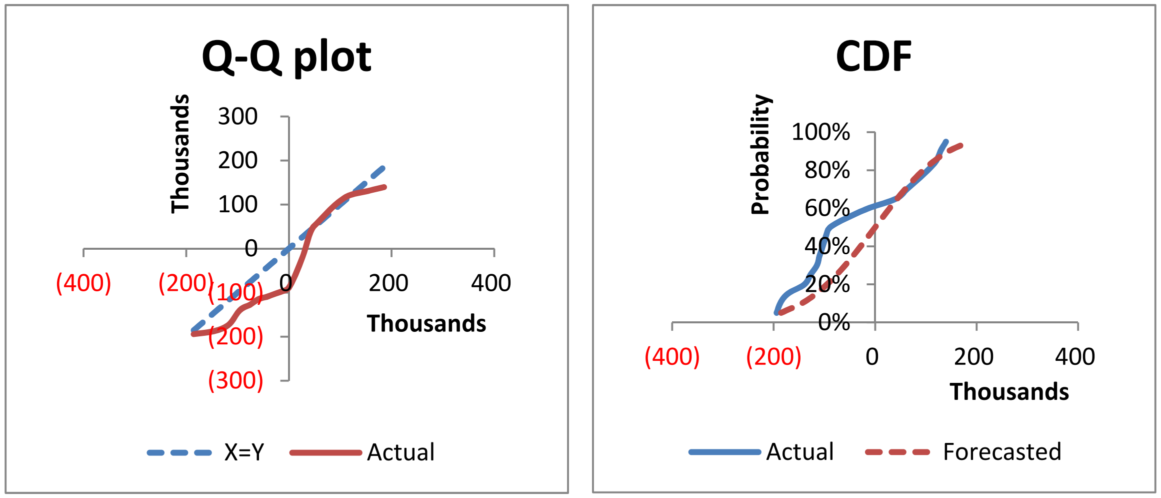


Figure 7-6 Actual contingency and forecasted contingency of Case 3 (input size: 262; forecast period: 129)

The forecasts are compared with actual contingencies using percentiles, as shown in Table 7-5. In both Case 1 and Case 3, the forecasts are very close to the actual contingencies: the actual 95th percentile contingencies were 4% more than the forecasted, which is equivalent to 0.3% of the average actual cost of the prediction period (half a year). In Case 2, the forecast is 16% less than the actual, equivalent to 1% of the actual cost of the average actual cost of the prediction period (a year). As stated by Flyvbjerg et al., the actual costs of tunnel and bridge projects are on average 34% higher than estimates, according to a statistical study on a sample of 258 transportation infrastructure projects worth US$90 billion (2002). Compared with cost overrun percentages in other tunnel projects, the contingency differences in all three cases are very minor.

Table 7-5 Cost contingency using annual or semi-annual data

Case	95 percentile of all actual contingency	Average of forecast at 95% confidence level	Amount Diff.	Diff. / Actual contingency	Average cost of forecast duration	Actual contingency / cost	Diff. / Actual cost
Case 1	-188,582	-181,912	-6,670	4%	-2,201,152	8.6%	0.3%
Case 2	-269,666	-312,882	43,216	-16%	-4,290,747	6.3%	-1.0%
Case 3	-194,193	-185,456	-8,737	4%	-2,060,101	9.4%	0.4%

7.5 Discussion

The VaR method provides a quantitative and efficient way to forecast and update necessary contingency. It analyzes a project's daily cost and daily earnings, instead of analyzing risks and their magnitudes. Risk leads to cost overruns, which is why projects need cost contingency. Although risk analysis helps us to understand various potential risks and their impacts before their occurrences, it is a challenge to properly identify and quantify them. If two related risks occur simultaneously, the impact magnitude is likely to be higher than the simple aggregate of their impacts when occurring individually. The impact due to multiple risks could weaken the validity of early risk analysis. However, this does not cause a problem in our model, because the model inputs rely on actual cost every day, the change of which provides a good gauge of risk effect.

The accuracy of the model presented in this research is not dependent on the original cost estimates or expert judgements, as in other contingency models (Lhee et al. 2009; Cioffi and Khamooshi 2009; Mak and Picken 2000; Sonmez et al. 2007; Mohamed et al. 2009). In the case study, the project estimate was made in 2006, and is used as a parameter in contingency calculation. Even though the estimate was made several years before the project construction, it does not affect the validity of the VaR method. If the actual cost turns out to be higher than the estimate, the project contingency will be correspondingly higher.

The method also provides a way of updating contingency by periods. Traditionally, cost contingency is estimated and included as a part of project cost estimation before the project commences. This research proposes to set it by periods, such as yearly periods, and update it at the end of each period, according to project properties and management style. This is an aid for updating the annual budget plan and resource utilization, especially for large companies that always have multiple projects on hand. Periodically updated cost contingencies also provide a chance to include the latest information and remove out-dated information. As demonstrated in our case study, we can update contingency forecasts for every year or half a year. In addition, the total project contingency can also be forecasted using this method by assuming project completion date, if needed.

One shortcoming of VaR is that a long period of historical data is required. When lacking data at the beginning of a project, we have to use historical data from other similar projects in modeling. The model reliability is constrained by the quality of historical data and the relevance of the data with the current project. Another common criticism of VaR is related to the unverified assumption that everything follows a multivariate Normal distribution. One caveat found by the author is that data collection, filtering, and analysis are very time-consuming when applying this method, particularly the determination of daily costs.

7.6 Applications

This section discusses the opportunities and challenges in applying the method to other projects. Firstly, the method remains applicable at different levels of control. This research applied the VaR method for project-level contingency forecasting for project control purposes. The method is shown to be practical and useful for high level budgeting and contingency planning, but could also be extended into lower levels, such as for different tasks, and into higher levels of a whole company or portfolio.

This method would be effective for projects of varying contract types, as it is used for contractors to calculate cost contingency. Contracts of varying types define the method and the amount of payment to contractors differently. Contract type affects cash flow, but not the actual cost and potential project earning, which are the inputs of VaR models. Therefore, this method is potentially applicable to all kinds of projects, regardless of contract type. It may seem that contractors are more concerned about contingency for lump sum projects, because they take full risks in cost, and are less concerned for “cost plus” projects. However, contingency focuses on the cost due to downside risks, not profit, and contractors must always have enough funding, including contingency, to maintain cash flow regardless of contract type.

The author recommends a possible application of this method in highway engineering, because highway agencies have a particular interest in cost overrun and contingency (Molenaar et al. 2009). Olumide et al. (2010) state that most state highway agencies (SHAs) applied one or more of the following three methods to decide contingency: 1) a predetermined contingency percentage for all projects; 2) a unique decision by individual estimators or project managers; or 3) a formal risk-analysis based method. For example, the Washington State Department of Transportation (WSDOT) developed and applied Cost Estimating Validation Process (CEVP) (Molenaar 2005). This VaR method as introduced in this research could potentially provide a new qualitative method for forecasting highway contingency and warrants further study.

Future advancements of the method could include automating data processing and improving model validity. This model could be implemented as a part of a project control system or ERP software, in order to simplify the process of data collection and processing. Also, as the project potential earning/loss may not be always normally distributed, the ability to dynamically change the distribution type for forecasting could also be useful for future research. Finally, this method can also be combined with simulation, which forecasts project duration and could be used to inform contingency calculation.

7.7 Conclusion

This research proposed a qualitative method based on VaR to forecast and update project contingencies at a given confidence level throughout project execution. The method minimizes human bias in risk assessment which is a common issue in most other contingency methods. The model can be continuously updated to generate new forecasts based on newly available information from daily activities. The method has been illustrated in a tunnel construction project based on actual cost records and productivity. The case study verification shows that the forecasted contingencies are very close to the actual, and that the forecast accuracy decreases when the prediction period increases. This method provides a chance to set and allocate appropriate contingencies on certain timelines, which improves budget planning and resource utilization for major companies and agencies which run multiple projects lasting for years. In addition, the paper also discussed the advantages, disadvantages, applicability, and future improvements of the VaR method.

CHAPTER 8 CONCLUSION

8.1 Research Summary

This research tries to improve project control as more project information is received as a project proceeds, because uncertainty and risk can be identified and decreased upon more available information, and the prediction for future performance and future work plan can be improved.

This research investigated effective tools and methods for dynamic and collaborative project control during project execution. The main research work refers to a project monitoring system for data collection and analysis, and information sharing, tunnel construction simulation, adaptive modeling, project forecasting and planning, and cost contingency forecasting and planning.

This research has addressed the following issues:

1) How to effectively collect pertinent productivity data and monitor project progress from various construction sites in light of the state of the art;

2) How to develop a simulation model for a construction project, which accounts for project logic, work sequence, resource constraints and uncertainty;

3) How to maintain valid model inputs to reflect the latest performance of the project through project execution and to provide long-term support;

4) How to feed actual productivity data into a simulation model and integrate them to update the model; and

5) How to dynamically forecast and make reasonable and reliable up-to-date project plans in terms of schedule/production and cost, especially cost contingency.

This research began with a web-based project monitoring system to support project progress data collection. It also analyzes project progress data by providing three different project progress reports, including daily report, weekly report, and monthly report. The collected progress data is integrated into a simulation model through a Bayesian updating simulation component by automatically updating model inputs according to project progress information collected from the project monitoring system. The simulation model was developed based on HLA, which is considered state of the art in discrete event simulation. Finally, the simulation model provides the time, labour hours, and equipment hours for completing the project. The labour hours and equipment hours serve as the basis of cost forecasting and budget allocation.

Budget allocation includes cost contingency to cover cost overrun due to risks and uncertainties. This research also introduced a quantitative method based on Value at Risk, and investigated its applicability in a tunnel construction project. The forecasts compare favourably with actual contingencies. The forecasted cost contingency using VaR, together with labour hours and equipment hours from the simulation model, provides the essence for project budget planning and allocation.

The overall goal of this research was to improve dynamic and collaborative project control. It investigated and developed project monitoring system, construction simulation model, and innovative methods for dynamic project planning, referring to production, cost, resource, and cost contingency. This research is demonstrated through a tunnel construction project, but the methodology implemented for improving project control is generic for all construction projects.

8.2 Research Contributions

The key contribution of this research is the successful integration of discrete event simulation, automated data collection, automated simulation model input updating, and the theory of Last Planner System, to achieve dynamic and collaborative project control. The discrete event simulation was developed based on High level Architecture, which represents the state of the art in construction simulation. The HLA-based simulation model enabled the integration of different techniques, as well as collaboration among project stakeholders.

Another contribution is timely, automated model input updating. The simulation model inputs can be automatically updated, according to actual daily project progress data collection. A Bayesian updating technique implemented in the simulation model and a web-based database application are developed to support the data collection and analysis on a daily basis.

This research further assisted dynamic and systematic planning. It produced project plans and/or schedules at different detail levels, from an overview of the entire project in a milestone plan, to daily activities and their resource and constraint requirements in a weekly work plan; duration varied from time to project completion, to weekly time constraints. As the HLA-based simulation model takes the construction sequence logic, resource requirement, and latest project progress data into account, it is able to support dynamic construction planning.

This research also proposed a qualitative method for forecasting and updating project contingencies throughout project execution. The method does not rely on human judgments for risk assessment, and utilized actual project cost to forecast contingency. It avoids a common weakness, human bias in risk analysis, which plagues most other contingency forecasting methods.

Additionally, during the work of this research, the following papers have been prepared for sharing the knowledge and experience with other researchers and practitioners in construction management:

1) Integrated project progress data into a simulation model has been documented in a paper titled “Integrating real time project progress input into a construction simulation model,” which is accepted in the proceedings of the Winter Simulation Conference in December, 2011.

2) An an integrated system approach was proposed and an HLA-based simulation model for dynamic project control through systematic and reliable planning was developed. Additionally, key inputs in the simulation model were continuously updated using Bayesian inference to assure valid modeling, and project plans at three different levels of detail were maintained in accordance with the theory of LPS. This research also investigated the applicability of using HLA concepts in simulation in achieving dynamic project control. All these have been documented in a paper titled, “A simulation-based integrated system for collaborative and dynamic project control,” which is in final preparation for submission for publication in the *Journal of Construction Engineering and Management*, ASCE.

3) A quantitative method for updating and planning cost contingency using actual project cost through project execution was introduced and successfully tested and verified into a tunnel construction project. This has been documented in a paper titled, “A quantitative method for updating cost contingency throughout project execution,” which is accepted in the *Journal of Construction Engineering and Management*, ASCE.

8.3 Industrial Contributions

The developed web-based project monitoring system can help to reduce the cost and time of data collection and improve the quality of data for modeling. It also facilitated data sharing and transfer between different project participants. The web-based solution enables data capture using PDAs such as the iPhone™, which eases the time and effort for collecting information from scattered construction sites. The project monitoring system also provides three types of project reports to provide insight analysis.

The simulation system can provide long-term support for the project management team during project execution because the simulation model is validated through continuous incorporation of actual project performance and updating model inputs. The integrated simulation system provides a powerful tool for planning project production and cost, and analyzing resource requirements in both long term and short term. The developed simulation system can also be used to try what-if scenario analysis, and investigate the effectiveness of corrective actions, which consequently help decision making in project control.

The cost contingency forecasting method can be used to set and allocate project budget along with project timelines, instead of holding all cost contingency at the beginning of a project. This is beneficial for major companies which run multiple projects lasting for years, allowing them to set appropriate contingencies on certain timelines and to improve resource utilization. All these tools and methods provide the potentials to improve the efficiency and accuracy of project control.

8.4 Recommendations for future research

The presented research can be extended and integrated with other research areas. The following are some of these areas:

1) Automated data collection. New technology in automated data collection, such as RFID, has been introduced to the construction industry to automate project progress data collection. It could reduce time and cost in data collection, reduce data entry errors, and improve data quality.

2) Investigating new techniques in updating model inputs. There are techniques, other than Bayesian inference as implemented in this research, such as time series, that could provide better forecasting.

3) Integrating VaR into the simulation system. VaR modeling method can be developed as a component in the HLA-based simulation system, which can be easily reused and applied for other construction projects.

4) Integrating project monitoring system with a company's financial management system, or other ERP systems. Cost and production data provide the basis for project control. This cost data is usually well documented in the company's financial management system, because the company has to rigorously record all indirect and direct actual cost for accounting purposes. One direct benefit for this research is reduced time and effort needed for implementing VaR method. The author of this research had to manually collect cost data and production data from different sources when applying VaR into a construction project, since both actual progress and actual cost are used for model inputs.

5) Investigating the relationship between crew size and productivity. A typical tunnel crew usually includes a foreman, a TBM operator, and a few other labourers. The actual daily productivity is influenced by the crew work attendance. This research assumed a standard crew and did not account the impact from attendance rate. It is recommended to explore this area using advanced computer techniques.

6) Exploiting the simulation system for assisting project management team in decision-making. An interface that allows a user to interact and intervene

in the simulation system and to try different decision variables is recommended. A well-designed interface can incorporate the expertise and judgment from project management professionals, thus improving the flexibility of the modeling system.

7) Enhance the simulation model for more tunnel construction methods. The proposed simulation model was based on the tunneling methods employed by the City of Edmonton, such as TBM tunneling. Modeling more tunneling methods will extend the function of the simulation system. It also allows users to experiment with different construction strategies and improve project efficiency.

Appendix A. Tunnel federation code

a. Shaft Federate

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Cosye.Framework;
using Cosye.Hla.Rti;
using Simphony.Mathematics;
using Simphony.Simulation;
using Cosye.Tunneling.TunnelFederate;

namespace Cosye.Tunneling.Shaft2
{
  public partial class ShaftFederateControl : FederateControl
    {
        #region Private Fields

        private        BasicDiscreteEventEngine        MyEngine        =        new
BasicDiscreteEventEngine();
      private ShaftInfo[] shaftInfos;
      private Size shaftInfoSize = new Size(290, 233);
      private Padding shaftInfoMargin = new Padding(3);
      private EventHandler shaftInfoCloseEvent;

      private Action<Shaft> pilingEvent;
      private Action<RectangularShaft> installWalerEvent;
      private Action<Shaft> excavateOneShaftSectionEvent;
      private Action<Shaft> ScheduleNextSectionOrShaftCompletionEvent;
      private Action<Shaft> finishPilingEvent;

      private Entity shiftControlEntity = new Entity();
      private DateTime startDateTime = DateTime.Today;
      private Action<Entity> ShiftOnEvent;
      private Action<Entity> ShiftOffEvent;
      private ShiftAwareEntities allEntities = new ShiftAwareEntities();
      private ShiftAwareEntities suspendedEntities = new ShiftAwareEntities();
      private Shift theShift;
      #endregion

        #region Constructor
        public ShaftFederateControl()
      {
        InitializeComponent();

            // Set up ShaftFederateControl to make future ShaftInfos based on the
ShaftInfo set up in the designer
        this.shaftInfoSize = this.shaftInfoTemplate.Size;
        this.shaftInfoMargin = this.shaftInfoTemplate.Margin;
        this.shaftInfoCloseEvent = shaftInfo_Close;

        // Note: for consistency, the EventHandlers of shaftInfoTemplate should
not be set to anything in the designer.
        this.shaftInfoTemplate.Close += this.shaftInfoCloseEvent;

        // Set up UI to be non-interactive until the simulation is running.
        this.shaftInfoTemplate.ShowRemoveButton = false;
        this.shaftInfoTemplate.Enabled = false;
        this.addButton.Enabled = false;

        // Initialize events
        this.pilingEvent = new Action<Shaft>(Piling);
        this.installWalerEvent = new Action<RectangularShaft>(InstallWaler);
        this.excavateOneShaftSectionEvent = new
Action<Shaft>(ExcavateOneShaftSection);
        this.ScheduleNextSectionOrShaftCompletionEvent = new
Action<Shaft>(ScheduleNextSectionOrShaftCompletion);

        this.finishPilingEvent = new Action<Shaft>(FinishPiling);
        this.ShiftOffEvent = new Action<Entity>(ShiftOffEventHandler);
        this.ShiftOnEvent = new Action<Entity>(ShiftOnEventHandler);

        this.MyEngine.InitializeEngine();
        }
        #endregion

        #region Private Methods

      #region Interface Methods

      #region private int GetNonNullShaftInfosCount()
      /// <summary>
      /// Gets the number of non-null shaftInfos in this.shaftInfos, assuming that
they are all at the beginning of the array.
      /// </summary>
      /// <returns>The number of non-null shaftInfos in this.shaftInfos</returns>
      private int GetNonNullShaftInfosCount()
```

```
        {
            int i = 0;
            while (i < this.shaftInfos.Length && this.shaftInfos[i] != null)
                i++;
            return i;
        }
        #endregion

        #region private void AddShaftInfo()
        /// <summary>
        /// Adds a new ShaftInfo to the user interface. This function should not be
called if the number of
        /// displayed ShaftInfos is already equal to the number of shafts in the
simulation.
        /// </summary>
        private void AddShaftInfo()
        {
            int newIndex = GetNonNullShaftInfosCount();

            // setup the new ShaftInfo to match the one that was set up in the designer.
            this.shaftInfos[newIndex] = new ShaftInfo();
            this.shaftInfos[newIndex].Size = this.shaftInfoSize;
            this.shaftInfos[newIndex].Close += this.shaftInfoCloseEvent;
                  this.shaftInfos[newIndex].CalendarFactory = this.calendarFactory;
                  this.shaftInfos[newIndex].CircularShaftFactory                          =
this.circularShaftFactory;
                  this.shaftInfos[newIndex].RectangularShaftFactory                      =
this.rectangularShaftFactory;
            this.Controls.Add(this.shaftInfos[newIndex]);

            this.shaftInfos[0].ShowRemoveButton = true;
            if (newIndex >= this.shaftInfos.Length - 1)
                this.addButton.Visible = false;

            DoLayout();
        }
        #endregion

        #region private void RemoveShaftInfo(ShaftInfo)
        /// <summary>
        /// Removes a ShaftInfo from the user interface and from this.shaftInfos
        /// </summary>
        /// <param name="toRemove"></param>
        private void RemoveShaftInfo(ShaftInfo toRemove)
        {
            // remove from the UI
            this.Controls.Remove(toRemove);

            // remove from the array
            for (int i = 0; i < this.shaftInfos.Length; i++)
            {
               if (this.shaftInfos[i] == toRemove)
               {
                  this.shaftInfos[i].Dispose();
                  this.shaftInfos[i] = null;
                  break;
               }
            }

            // ensure that all displayed ShaftInfos are at the beginning of the array
            TidyArray(this.shaftInfos);

            // if a ShaftInfo was removed, it must be possible to add it back in, so...
            this.addButton.Visible = true;

            // if there is only one shaft left, don't let the user remove it too
            if (GetNonNullShaftInfosCount() == 1)
               this.shaftInfos[0].ShowRemoveButton = false;

            DoLayout();
         }
         #endregion

         #region private void DoLayout()
         /// <summary>
         /// Lays out this form's controls left to right, top to bottom.
         /// The controls must be in order in this.shaftInfo, and this.shaftInfo must
match the shaftInfos on this form.
         /// </summary>
         private void DoLayout()
         {
            // get shaftInfo total dimensions
            int totalWidth = this.shaftInfoSize.Width + this.shaftInfoMargin.Left +
this.shaftInfoMargin.Right;
            int totalHeight = this.shaftInfoSize.Height + this.shaftInfoMargin.Top +
this.shaftInfoMargin.Bottom;

            // layout ShaftInfos
            Point location = new Point(0, 0);
            for (int i = 0; i < this.shaftInfos.Length; i++)
            {
               if (this.shaftInfos[i] != null)
            {
                location = new Point(((i*totalWidth) % Math.Max(1, this.Width)) +
this.shaftInfoMargin.Left, (i / Math.Max(1, (this.Width / Math.Max(1,
Math.Min(this.Width, totalWidth)))) * totalHeight) + this.shaftInfoMargin.Top);
                this.shaftInfos[i].Location = location;
            }
        }

        // place button
        if (location.X + this.shaftInfoSize.Width + this.shaftInfoMargin.Right +
this.addButton.Margin.Left + this.addButton.Width +
this.addButton.Margin.Right > this.Width)
        {
            // place button on next row
            this.addButton.Location = new Point(this.addButton.Margin.Left,
location.Y + this.shaftInfoSize.Height + this.addButton.Margin.Top);
        }
        else
        {
            // place button at the end of this row
            this.addButton.Location = new Point(location.X +
this.shaftInfoSize.Width + this.shaftInfoMargin.Right +
this.addButton.Margin.Left, location.Y - this.shaftInfoMargin.Top +
this.addButton.Margin.Top);
        }
    }
    #endregion

    #region private void TidyArray(object[] array)
    /// <summary>
    /// moves all non-null entries in an array to the beginning of the array without
changing their order.
    /// </summary>
    /// <param name="array">The array to tidy.</param>
    private void TidyArray(object[] array)
    {
        int arrayLength = array.Length;
        bool passWithNoSwaps = false;

        for (int i = 0; i < (arrayLength - 1); i++)
        {
            passWithNoSwaps = true;
            for (int j = 0; j < (arrayLength - 1 - i); j++)
            {
                if ((array[j] == null) && (array[j + 1] != null))
                {
                    array[j] = array[j + 1];
                    array[j + 1] = null;
                    passWithNoSwaps = false;
                }
            }

            if (passWithNoSwaps)    // array is already tidy
                return;
        }
    }
    #endregion

        #region private void OnResize(object, EventArgs)
        private void OnResize(object sender, EventArgs e)
    {
        DoLayout();
        }
        #endregion

        #endregion

        #region Calculator Methods
        // These methods just calculate stuff, they shouldn't update any simulation
instance attributes

    #region private double TotalPilingTime(Shaft)
    /// <summary>
    /// Calculates the total time to drive all the piles for a shaft.
    /// </summary>
    /// <param name="shaft">The shaft to calculate the total piling time
for.</param>
    /// <returns>The total piling time for the shaft, in seconds.</returns>
    private double TotalPilingTime(Shaft shaft)
    {
        int pileQuantity = shaft.PileQuantity;

        if (pileQuantity < 0)
        {
            // PileQuantity values less than 0 indicate that the quantity should be
calculated from the dimensions.
            // The calculations are done here:
            if (shaft.Shape == ShaftShape.Rectangular)
            {
                RectangularShaft rectShaft = (RectangularShaft)shaft;
                if (rectShaft.PileWidth != 0)
                pileQuantity = 4 * (Convert.ToInt32((rectShaft.ExcavationLength
+ rectShaft.ExcavationWidth) / (1.7 * shaft.PileWidth)) - 1);
            }
            else
            {
                CircularShaft circShaft = (CircularShaft)shaft;
                if (circShaft.PileWidth != 0)
                    pileQuantity = Convert.ToInt32(Math.PI *
circShaft.ExcavationDiameter * 0.3048 / circShaft.PileWidth);
            }
        }

        double totalTime = 0;
        for (int i = 0; i < pileQuantity; i++)
                totalTime += shaft.OnePileDrivingDuration.Sample();

        return totalTime;
    }
    #endregion

    #region private double GetSectionExcavationTime(ShaftSection,
ExcavationMethod)
    /// <summary>
    /// Calculates the excavation time for the part of a shaft section done with a
particular method (hand or machine).
    /// </summary>
    /// <param name="section">The shaft section to get the partial excavation
time of</param>
    /// <param name="method">The excavation method of the part of the shaft
section to get the excavation time of</param>
    /// <returns>The time, in seconds, to excavate the part of a shaft section that
is done with the given excavation method.</returns>
    private double GetSectionExcavationTime(ShaftSection section,
ExcavationMethod method)
    {
        // get the dirt volume
        double dirtVolume = GetSectionExcavatedDirtVolume(section, method);
        // use this function's overload to get the time.
            return GetSectionExcavationTime(section, method, dirtVolume);
    }
    #endregion

    #region private double GetSectionExcavationTime(double,
ExcavationMethod)
    /// <summary>
    /// Calculates the excavation time for a volume of dirt excavated with a
particular method (hand or machine).
    /// </summary>
    /// <param name="excavatedDirtVolume">The volume of dirt excavated
with the given method.</param>
    /// <param name="method">The excavation method used to excavate the
dirt</param>
    /// <returns>The time, in seconds, to excavate the given volume of dirt with
the given excavation method.</returns>
      private double GetSectionExcavationTime(ShaftSection section,
ExcavationMethod method, double excavatedDirtVolume)
    {
      if (method == ExcavationMethod.Machine)
        return excavatedDirtVolume /
section.MachineExcavationRate.Sample();
      else
            return excavatedDirtVolume /
section.HandExcavationRate.Sample();
    }
    #endregion

    #region private double GetSectionExcavatedDirtVolume(ShaftSection,
ExcavationMethod)
    /// <summary>
    /// Calculates the volume of dirt excavated from the part of a shaft section
    /// that is excavated with a particular method (hand or machine).
    /// </summary>
    /// <param name="section">The shaft section to get the partial excavated dirt
volume of</param>
    /// <param name="method">The excavation method of the part of the shaft
section to get the excavated dirt volume of</param>
    /// <returns>The volume of dirt, in cubic meters, excavated from the part of
the shaft section that is done with the given excavation method.</returns>
    private double GetSectionExcavatedDirtVolume(ShaftSection section,
ExcavationMethod method)
    {
      // get the shaft cross-sectional area
      return section.GetVolume(method) * section.SoilSwellFactor;
    }
    #endregion

    #region private double InstallWalerTime(RectangularShaft)
    /// <summary>
    /// Calculates the time to install a waler in a rectangular shaft.
    /// </summary>
```

```
        /// <param name="theShaft">The shaft to calculate the waler installation
time for.</param>
        /// <returns>The installation time of the waler, in seconds</returns>
        private double InstallWalerTime(RectangularShaft theShaft)
        {
           //return 5 * (theShaft.ExcavationWidth + theShaft.ExcavationLength) /
15.5 * 60 * 60;
           return 18000.0 * (theShaft.ExcavationWidth + theShaft.ExcavationLength)
/ 15.5;
        }
        #endregion

        #region calculateDateTime(double)
        private DateTime calculateDateTime(double logicTime)
        {
           return this.startDateTime.AddSeconds(logicTime);
        }
        #endregion
        #endregion

        #region Simulation Methods
        // These methods update simulation instance attributes

        #region StartSimulation(Shaft)
        /// <summary>
        /// Begins the simulation of an unstarted shaft.
        /// </summary>
        /// <param name="shaft">The shaft to begin simulating.</param>
        private void StartSimulation(Shaft shaft)
        {
           switch (shaft.State)
           {
              case ShaftState.Unstarted:
                 // shaft hasn't started yet,
                 // set excavated dirt volume to 0 and
                 // schedule the first part of the shaft construction process: piling

shaft.AttributeOwnershipAcquisition(shaftAttributes.ExcavatedDirtVolume,
shaftAttributes.State);

    shaft.UpdateAttributeValues(shaftAttributes.ExcavatedDirtVolume, 0);
                      if (shaft.StartTime > this.fedAmb.CurrentTime +
this.fedAmb.Lookahead)
                 shaft.UpdateAttributeValues(shaft.StartTime, shaftAttributes.State,
ShaftState.Piling);
                 else
```

```
                        shaft.UpdateAttributeValues(shaftAttributes.State,
ShaftState.Piling);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.ExcavatedDirt
Volume, shaftAttributes.State);

            this.MyEngine.ScheduleEvent(shaft, this.pilingEvent, shaft.StartTime
- this.MyEngine.TimeNow);
                    break;
        }
    }
    #endregion

    //#region ScheduleNextSectionOrShaftCompletion(Shaft, double)
    ///// <summary>
    ///// Takes a shaft that will complete its current section at a given time,
    ///// and schedules the completion of the current section, and either the
    ///// completion of the shaft or the start of the next section.
    ///// </summary>
    ///// <param name="shaft">The shaft whose current section will be
completed at the end of the duration.</param>
    ///// <param name="duration">The duration after which the current shaft
section will be complete, measured from this.eventEngine.TimeNow</param>
    //private void ScheduleNextSectionOrShaftCompletion(Shaft shaft, double
duration)
    //{
    //    // get the current section from the correct ShaftSectionFactory
    //    ShaftSection currentSection = shaft.GetCurrentSection();

    //    // let the federation know when the current section will be finished
    //
currentSection.AttributeOwnershipAcquisition(shaftSectionAttributes.State,
shaftSectionAttributes.FinishTime);
    //    List<ObjectAttributeValuePair<ShaftSection>> sectionFinishUpdates =
new List<ObjectAttributeValuePair<ShaftSection>>(2);
    //                sectionFinishUpdates.Add(shaftSectionAttributes.State,
ShaftSectionState.Finished);
    //            sectionFinishUpdates.Add(shaftSectionAttributes.FinishTime,
this.MyEngine.TimeNow + duration);
    //        currentSection.UpdateAttributeValues(this.MyEngine.TimeNow +
duration, sectionFinishUpdates);
    //
currentSection.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttribut
es.State, shaftSectionAttributes.FinishTime);

    //    // if this is not the last section
```

```
        //    if (currentSection.NextSection != ObjectInstanceHandle.Empty)
        //    {
        //        // let the federation know when the shaft's current section will change
        //        // to the next section
        //        shaft.AttributeOwnershipAcquisition(shaftAttributes.CurrentSection);
        //            shaft.UpdateAttributeValues(this.MyEngine.TimeNow + duration,
shaftAttributes.CurrentSection, currentSection.NextSection);
        //
shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.CurrentSection
);

        //        // get a reference to the next section fromth e correct factory
        //        ShaftSection nextSection = currentSection.GetNextSection();

        //        // let the federation know when the next section will be started
        //
nextSection.AttributeOwnershipAcquisition(shaftSectionAttributes.State,
shaftSectionAttributes.StartTime);
        //         nextSection.UpdateAttributeValues(shaftSectionAttributes.StartTime,
this.MyEngine.TimeNow + duration);
        //        ShaftSectionState nextSectionState;
        //                              if     (GetSectionExcavationTime(nextSection,
ExcavationMethod.Machine) > 0)
        //            nextSectionState = ShaftSectionState.MachineExcavation;
        //        else
        //            nextSectionState = ShaftSectionState.HandExcavation;
        //               nextSection.UpdateAttributeValues(this.MyEngine.TimeNow  +
duration, shaftSectionAttributes.State, nextSectionState);
        //
nextSection.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttributes.
State, shaftSectionAttributes.StartTime);

        //        // schedule the excavation event for the next section
        //                                              this.MyEngine.ScheduleEvent(shaft,
this.excavateOneShaftSectionEvent, duration);
        //    }
        //    else // this is the last section of the shaft
        //    {
        //        // let the federation know when the shaft will be complete
        //              shaft.AttributeOwnershipAcquisition(shaftAttributes.FinishTime,
shaftAttributes.CurrentSection, shaftAttributes.State);
        //                       shaft.UpdateAttributeValues(shaftAttributes.FinishTime,
this.MyEngine.TimeNow + duration);
        //           List<ObjectAttributeValuePair<Shaft>> shaftCompletionUpdates =
new List<ObjectAttributeValuePair<Shaft>>(2);
```

```
    //                 shaftCompletionUpdates.Add(shaftAttributes.CurrentSection,
ObjectInstanceHandle.Empty);
    //                           shaftCompletionUpdates.Add(shaftAttributes.State,
ShaftState.Finished);
    //           shaft.UpdateAttributeValues(this.MyEngine.TimeNow + duration,
shaftCompletionUpdates);
    //
shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.FinishTime,
shaftAttributes.CurrentSection, shaftAttributes.State);
    //   }
    //}
    //#endregion

    #endregion

    #endregion

    #region Events

    #region Simulation Events

    #region private void Piling(Shaft, EventArgs)
    /// <summary>
    /// Simulates the piling event, scheduling the start of excavation after piling is
finished.
    /// </summary>
    /// <param name="shaft">An unstarted shaft to begin piling at
this.eventEngine.TimeNow</param>
    /// <param name="e"></param>
    private void Piling(Shaft shaft)
    {
      // calculate total piling time. This is also the start time of the first shaft
section,
      // relative to this.eventEngine.TimeNow, of course.
      double duration = TotalPilingTime(shaft);
          this.MyEngine.ScheduleEvent(shaft,              this.finishPilingEvent,
Math.Max(duration - fedAmb.Lookahead, 0.0));
      //// let the federation know when the current shaft will finish piling and
start it's first section
      //shaft.AttributeOwnershipAcquisition(shaftAttributes.PilingFinishTime,
shaftAttributes.State);
      //shaft.UpdateAttributeValues(shaftAttributes.PilingFinishTime,
this.MyEngine.TimeNow + duration);
      //shaft.UpdateAttributeValues(this.MyEngine.TimeNow      +      duration,
shaftAttributes.State, ShaftState.Excavating);
```

```
//shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.PilingFinishT
ime, shaftAttributes.State);
            //if (shaft.CurrentSection == ObjectInstanceHandle.Empty)
            //{
            //    shaft.AttributeOwnershipAcquisition(shaftAttributes.CurrentSection);
            //        shaft.UpdateAttributeValues(this.MyEngine.TimeNow + duration,
shaftAttributes.CurrentSection, shaft.FirstSection);
            //
shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.CurrentSection
);
            //}

            //ShaftSection firstSection = shaft.GetFirstSection();
            //// let the federation know when the first section excavation will start.
            //firstSection.AttributeOwnershipAcquisition(shaftSectionAttributes.State,
shaftSectionAttributes.StartTime);
            //firstSection.UpdateAttributeValues(shaftSectionAttributes.StartTime,
this.MyEngine.TimeNow + duration);
            //double                    machineExcavationDuration                    =
GetSectionExcavationTime(firstSection, ExcavationMethod.Machine);
            //if (machineExcavationDuration > 0)
            //            firstSection.UpdateAttributeValues(this.MyEngine.TimeNow   +
duration, shaftSectionAttributes.State, ShaftSectionState.MachineExcavation);
            //else
            //            firstSection.UpdateAttributeValues(this.MyEngine.TimeNow   +
duration, shaftSectionAttributes.State, ShaftSectionState.HandExcavation);

//firstSection.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttributes
.State, shaftSectionAttributes.StartTime);

            //// schedule the next event: excavation of the first section
            //this.MyEngine.ScheduleEvent(shaft,  this.excavateOneShaftSectionEvent,
duration);
        }
        #endregion

        #region private void FinishPiling(Shaft, EventArgs)
        /// <summary>
        /// Finalize piling event, update attributes, and scheduling the start of
excavation after piling is finished.
        /// </summary>
        ///   <param   name="shaft">An   unstarted   shaft   to   begin   piling   at
this.eventEngine.TimeNow</param>
        /// <param name="e"></param>
        private void FinishPiling(Shaft shaft)
        {
          double duration = fedAmb.Lookahead;
          // let the federation know when the current shaft will finish piling and start
it's first section
          shaft.AttributeOwnershipAcquisition(shaftAttributes.PilingFinishTime,
shaftAttributes.State);
          shaft.UpdateAttributeValues(shaftAttributes.PilingFinishTime,
this.MyEngine.TimeNow + duration);
          shaft.UpdateAttributeValues(this.MyEngine.TimeNow    +    duration,
shaftAttributes.State, ShaftState.Excavating);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.PilingFinishTi
me, shaftAttributes.State);
          if (shaft.CurrentSection == ObjectInstanceHandle.Empty)
          {
            shaft.AttributeOwnershipAcquisition(shaftAttributes.CurrentSection);
            shaft.UpdateAttributeValues(this.MyEngine.TimeNow    +    duration,
shaftAttributes.CurrentSection, shaft.FirstSection);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.CurrentSection
);
          }

          ShaftSection firstSection = shaft.GetFirstSection();
          // let the federation know when the first section excavation will start.
          firstSection.AttributeOwnershipAcquisition(shaftSectionAttributes.State,
shaftSectionAttributes.StartTime);
          firstSection.UpdateAttributeValues(shaftSectionAttributes.StartTime,
this.MyEngine.TimeNow + duration);

          double            machineExcavationDuration            =
GetSectionExcavationTime(firstSection, ExcavationMethod.Machine);
          if (machineExcavationDuration > 0)
            firstSection.UpdateAttributeValues(this.MyEngine.TimeNow       +
duration, shaftSectionAttributes.State, ShaftSectionState.MachineExcavation);
          else
            firstSection.UpdateAttributeValues(this.MyEngine.TimeNow       +
duration, shaftSectionAttributes.State, ShaftSectionState.HandExcavation);

firstSection.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttributes.
State, shaftSectionAttributes.StartTime);

          // schedule the next event: excavation of the first section
          this.MyEngine.ScheduleEvent(shaft,  this.excavateOneShaftSectionEvent,
duration);
        }
        #endregion

        #region private void ExcavateOneShaftSection(Shaft, EventArgs)
        /// <summary>
        /// Simulates the complete excavation process of the current section of a shaft,
        /// including machine and hand excavaation, and schedules the appropriate
next event
        /// when excavation is complete.
        /// </summary>
        /// <param name="shaft">The shaft to simulate the current section
excavation of</param>
        private void ExcavateOneShaftSection(Shaft shaft)
        {
            // ensure that the shaft has a "current section" before excavating
            if (shaft.CurrentSection == ObjectInstanceHandle.Empty)
            {
                shaft.AttributeOwnershipAcquisition(shaftAttributes.CurrentSection);
                shaft.UpdateAttributeValues(shaftAttributes.CurrentSection,
shaft.FirstSection);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.CurrentSection
);
            }

            ShaftSection section = shaft.GetCurrentSection();
            // calculate all of the excavation dirt volumes and durations
            double machineExcavatedDirtVolume =
GetSectionExcavatedDirtVolume(section, ExcavationMethod.Machine);
                double machineExcavationDuration =
GetSectionExcavationTime(section, ExcavationMethod.Machine,
machineExcavatedDirtVolume);
            double handExcavatedDirtVolume =
GetSectionExcavatedDirtVolume(section, ExcavationMethod.Hand);
                double handExcavationDuration = GetSectionExcavationTime(section,
ExcavationMethod.Hand, handExcavatedDirtVolume);
                double currentExcavatedDirtVolume = shaft.ExcavatedDirtVolume;

                // update shaft excavated dirt volume and depth after the durations

shaft.AttributeOwnershipAcquisition(shaftAttributes.ExcavatedDirtVolume,
shaftAttributes.CurrentDepth);
                List<ObjectAttributeValuePair<Shaft>> excavationUpdates = new
List<ObjectAttributeValuePair<Shaft>>(3);
                excavationUpdates.Add(shaftAttributes.ExcavatedDirtVolume,
currentExcavatedDirtVolume + machineExcavatedDirtVolume);
```

```
            if (this.MyEngine.TimeNow + machineExcavationDuration >
this.fedAmb.CurrentTime + this.fedAmb.Lookahead)
            {
                shaft.UpdateAttributeValues(this.MyEngine.TimeNow +
machineExcavationDuration, excavationUpdates);
                excavationUpdates.Clear();
            }
        excavationUpdates.Add(shaftAttributes.ExcavatedDirtVolume,
currentExcavatedDirtVolume + machineExcavatedDirtVolume +
handExcavatedDirtVolume);
            excavationUpdates.Add(shaftAttributes.CurrentDepth,
shaft.CurrentDepth + section.Depth);
            shaft.UpdateAttributeValues(this.MyEngine.TimeNow +
machineExcavationDuration + handExcavationDuration, excavationUpdates);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.ExcavatedDirt
Volume, shaftAttributes.CurrentDepth);

            // update scenario material cost
            Scenario scenario = shaft.GetScenario();

    scenario.AttributeOwnershipAcquisition(scenarioAttributes.MaterialCost);
            scenario.UpdateAttributeValues(this.MyEngine.TimeNow +
machineExcavationDuration + handExcavationDuration,
scenarioAttributes.MaterialCost, scenario.MaterialCost +
section.MaterialCostPerMeter * section.Depth);

    scenario.UnconditionalAttributeOwnershipDivestiture(scenarioAttributes.Mat
erialCost);

        // update section values
        if (handExcavatedDirtVolume > 0 && this.MyEngine.TimeNow +
machineExcavationDuration > this.fedAmb.CurrentTime +
this.fedAmb.Lookahead)
        {
          // if the lookahead allows us to set the ShaftSectionState to
HandExcavation when
                // the machine excavation is finished, do so.

    section.AttributeOwnershipAcquisition(shaftSectionAttributes.State);
                section.UpdateAttributeValues(this.MyEngine.TimeNow +
machineExcavationDuration, shaftSectionAttributes.State,
ShaftSectionState.HandExcavation);

    section.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttributes.S
tate);
```

```
            }
            if (shaft.Shape == ShaftShape.Rectangular && section.NextSection != 
ObjectInstanceHandle.Empty)
            {
                     // if the shaft is rectangular, install a waler at the end of excavating 
each section except the last one

     section.AttributeOwnershipAcquisition(shaftSectionAttributes.State);
                     section.UpdateAttributeValues(this.MyEngine.TimeNow + 
machineExcavationDuration + handExcavationDuration, 
shaftSectionAttributes.State, ShaftSectionState.InstallingWaler);

     section.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttributes.S
tate);

                this.MyEngine.ScheduleEvent((RectangularShaft)shaft, 
this.installWalerEvent, machineExcavationDuration + handExcavationDuration);
            }
            else // the shaft is non-rectangular, or it is finished.
            {
                //ScheduleNextSectionOrShaftCompletion(shaft, 
machineExcavationDuration + handExcavationDuration);
                this.MyEngine.ScheduleEvent(shaft, 
ScheduleNextSectionOrShaftCompletionEvent, machineExcavationDuration + 
handExcavationDuration);
            }
        }
        #endregion

        #region private void InstallWaler(RectangularShaft, EventArgs)
        /// <summary>
        /// Simulates the installation of a waler in a rectangular shaft,
        /// schedules the appropriate next event after the waler is installed.
        /// </summary>
        /// <param name="shaft">The rectangular shaft to install a waler in.</param>
        /// <param name="e"></param>
        private void InstallWaler(RectangularShaft shaft)
        {
            // Get the duration of the "install waler" process.
            double duration = InstallWalerTime(shaft);

            // schedule the correct next event
            this.MyEngine.ScheduleEvent(shaft, 
ScheduleNextSectionOrShaftCompletionEvent, duration);
        }
        #endregion
        #region private void ScheduleNextSectionOrShaftCompletion(Shaft)
        /// <summary>
        /// Takes a shaft that will complete its current section at a given time,
        /// and schedules the completion of the current section, and either the
        /// completion of the shaft or the start of the next section.
        /// </summary>
        /// <param name="shaft">The shaft whose current section will be completed
at the end of the duration.</param>
        /// <param name="duration">The duration after which the current shaft
section will be complete, measured from this.eventEngine.TimeNow</param>
        private void ScheduleNextSectionOrShaftCompletion(Shaft shaft)
        {
            // get the current section from the correct ShaftSectionFactory
            ShaftSection currentSection = shaft.GetCurrentSection();

            // let the federation know when the current section will be finished

currentSection.AttributeOwnershipAcquisition(shaftSectionAttributes.State,
shaftSectionAttributes.FinishTime);
            List<ObjectAttributeValuePair<ShaftSection>> sectionFinishUpdates =
new List<ObjectAttributeValuePair<ShaftSection>>(2);
            sectionFinishUpdates.Add(shaftSectionAttributes.State,
ShaftSectionState.Finished);
            sectionFinishUpdates.Add(shaftSectionAttributes.FinishTime,
this.MyEngine.TimeNow + this.fedAmb.Lookahead);
            currentSection.UpdateAttributeValues(this.MyEngine.TimeNow              +
this.fedAmb.Lookahead, sectionFinishUpdates);

currentSection.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttribut
es.State, shaftSectionAttributes.FinishTime);

            // if this is not the last section
            if (currentSection.NextSection != ObjectInstanceHandle.Empty)
            {
                // let the federation know when the shaft's current section will change
                // to the next section
                shaft.AttributeOwnershipAcquisition(shaftAttributes.CurrentSection);
                shaft.UpdateAttributeValues(this.MyEngine.TimeNow                   +
this.fedAmb.Lookahead,                          shaftAttributes.CurrentSection,
currentSection.NextSection);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.CurrentSection
);

                // get a reference to the next section fromth e correct factory
                ShaftSection nextSection = currentSection.GetNextSection();

                // let the federation know when the next section will be started

nextSection.AttributeOwnershipAcquisition(shaftSectionAttributes.State,
shaftSectionAttributes.StartTime);
                nextSection.UpdateAttributeValues(shaftSectionAttributes.StartTime,
this.MyEngine.TimeNow + this.fedAmb.Lookahead);
                ShaftSectionState nextSectionState;
                if                              (GetSectionExcavationTime(nextSection,
ExcavationMethod.Machine) > 0)
                    nextSectionState = ShaftSectionState.MachineExcavation;
                else
                    nextSectionState = ShaftSectionState.HandExcavation;
                nextSection.UpdateAttributeValues(this.MyEngine.TimeNow              +
this.fedAmb.Lookahead, shaftSectionAttributes.State, nextSectionState);

nextSection.UnconditionalAttributeOwnershipDivestiture(shaftSectionAttributes.
State, shaftSectionAttributes.StartTime);

                // schedule the excavation event for the next section
                this.MyEngine.ScheduleEvent(shaft,
this.excavateOneShaftSectionEvent, 0);
            }
            else // this is the last section of the shaft
            {
                // let the federation know when the shaft will be complete
                shaft.AttributeOwnershipAcquisition(shaftAttributes.FinishTime,
shaftAttributes.CurrentSection, shaftAttributes.State);
                shaft.UpdateAttributeValues(shaftAttributes.FinishTime,
this.MyEngine.TimeNow + this.fedAmb.Lookahead);
                List<ObjectAttributeValuePair<Shaft>>     shaftCompletionUpdates     =
new List<ObjectAttributeValuePair<Shaft>>(2);
                shaftCompletionUpdates.Add(shaftAttributes.CurrentSection,
ObjectInstanceHandle.Empty);
                shaftCompletionUpdates.Add(shaftAttributes.State,
ShaftState.Finished);
                shaft.UpdateAttributeValues(this.MyEngine.TimeNow              +
this.fedAmb.Lookahead, shaftCompletionUpdates);

shaft.UnconditionalAttributeOwnershipDivestiture(shaftAttributes.FinishTime,
shaftAttributes.CurrentSection, shaftAttributes.State);
            }
        }
        #endregion
```

```
# region ShiftOnEventHandler
/// <summary>
/// try to resume all suspended entities when shift changes to ON
/// </summary>
/// <param name="entity"></param>
private void ShiftOnEventHandler(Entity entity)
{
   this.suspendedEntities.ResumeEntities(MyEngine);
   this.suspendedEntities.UnionWith(this.allEntities);
}
#endregion

# region ShiftOffEventHandler
/// <summary>
/// try to suspend all scheduled entities when shift changes to OFF
/// </summary>
/// <param name="entity"></param>
private void ShiftOffEventHandler(Entity entity)
{
   this.suspendedEntities.SuspendEntitiesIfNeed(MyEngine);
}
#endregion

#endregion

#region Private Designer Events

#region private void fedAmb_BeginExecution(object, EventArgs)
/// <summary>
/// Starts this.eventEngine, sets up the one displayed ShaftInfo control,
   /// and begins the simulation of all shafts.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void fedAmb_BeginExecution(object sender, EventArgs e)
{
   // start the local event engine
   this.MyEngine.InitializeScenario();

   // get the total number of shafts so that we know what size to use for the
relevant arrays.
   int         shaftCount        =        this.circularShaftFactory.Count        +
this.rectangularShaftFactory.Count;

   if (shaftCount > 0)
   {
```

```
            // loop through all of the shafts and start simulating them.
            // Also add their names to this.shaftInfoItems, to be displayed in the
ShaftInfo combo-boxes
            foreach (CircularShaft shaft in this.circularShaftFactory)
               StartSimulation(shaft);
            foreach (RectangularShaft shaft in this.rectangularShaftFactory)
               StartSimulation(shaft);

            // set up the only displayed ShaftInfo to display the first shaft
            this.shaftInfos = new ShaftInfo[shaftCount];
            this.shaftInfoTemplate.Enabled = true;
            this.shaftInfos[0] = this.shaftInfoTemplate;

            this.addButton.Enabled = true;
            if (shaftCount == 1)
               this.addButton.Visible = false;

            this.Resize += new System.EventHandler(OnResize);
         }
         else
         {
            // there are no shafts to simulate,
            // change the user interface to reflect this.
            this.shaftInfoTemplate.Visible = false;
            this.addButton.Visible = false;
            Label noShafts = new Label();
            noShafts.AutoSize = true;
            noShafts.Text = "There are no shafts to simulate.";
            noShafts.Location = new Point(3, 3);
            this.Controls.Add(noShafts);
         }
      }
      #endregion

      #region private void fedAmb_TimeAdvanceGrant(object,
TimeAdvanceGrantEventArgs)
      private void fedAmb_TimeAdvanceGrant(object sender,
TimeAdvanceGrantEventArgs e)
      {
            //Process any internal events that should occur at the current time.
         this.MyEngine.Simulate(e.theTime);

            //Advance time to the time of the next internal event or attribute update
         rtiAmb.NextMessageRequest(Math.Min(this.MyEngine.TimeNext,
this.fedAmb.NextLocalAttributeUpdateTime));
      }
```

```
        #endregion

        #region private void fedAmb_EndExecution(object, EventArgs)
        private void fedAmb_EndExecution(object sender, EventArgs e)
        {
            this.MyEngine.FinalizeScenario();
        }
        #endregion

        #region private void shaftInfo_Close(object, EventArgs)
        private void shaftInfo_Close(object sender, EventArgs e)
        {
            RemoveShaftInfo((ShaftInfo)sender);
        }
        #endregion

        #region private void addButton_Click(object, EventArgs)
        private void addButton_Click(object sender, EventArgs e)
        {
            AddShaftInfo();
        }
        #endregion

        #region rectangularShaftFactory_DiscoverObjectInstance(object,
DiscoverObjectInstanceEventArgs)
        private void rectangularShaftFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
        {
            this.allEntities.Add(this.rectangularShaftFactory[e.theObject]);
        }
        #endregion

        #region circularShaftFactory_DiscoverObjectInstance(object,
DiscoverObjectInstanceEventArgs)
        private void circularShaftFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
        {
            this.allEntities.Add(this.circularShaftFactory[e.theObject]);
        }
        #endregion

        private void myShiftFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
        {
            var shift = myShiftFactory[e.theObject];
            if (e.theValues.Contains(shiftAttributes.IsWorking.GetAttributeHandle()))
```

```
          {
            if (e.theTime > 0)
            {
              double          timeOfUpdate          =          Math.Max(e.theTime,
this.rtiAmb.QueryLogicalTime() + fedAmb.Lookahead);
              if (shift.IsWorking)
              {
                MyEngine.ScheduleEvent(this.shiftControlEntity,    ShiftOnEvent,
e.theTime - MyEngine.TimeNow);
              }
              else
              {
                MyEngine.ScheduleEvent(this.shiftControlEntity,    ShiftOffEvent,
e.theTime - MyEngine.TimeNow);
              }
            }
          }
        }

        private    void    myShiftFactory_DiscoverObjectInstance(object    sender,
DiscoverObjectInstanceEventArgs e)
        {
          this.theShift = this.myShiftFactory[e.theObject];
        }

        #endregion

        #endregion
    }
}
```

b. Tunnel Federate I

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Cosye.Framework;
using Cosye.Hla.Rti;
using Simphony.Mathematics;
using Simphony.Simulation;
using Cosye.Tunneling.TunnelFederate;

namespace Cosye.Tunneling.Tunnel2
{
   public partial class TunnelFederateControl : FederateControl
   {
        private    readonly    BasicDiscreteEventEngine    eventEngine    =    new
BasicDiscreteEventEngine();

        private Action<TailTunnel> constructTailTunnelEvent;
        private Action<Undercut> installTbmEvent;
        private Action<TailTunnel> installSwitchEvent;
        private Action<Entity> checkCompletionEvent;

      private Entity shiftControlEntity = new Entity();
      private DateTime startDateTime = DateTime.Today;
      private Action<Entity> ShiftOnEvent;
      private Action<Entity> ShiftOffEvent;
      private ShiftAwareEntities allEntities = new ShiftAwareEntities();
      private ShiftAwareEntities suspendedEntities = new ShiftAwareEntities();
      private Shift theShift;
      private Calendar theCalendar;
      private Entity checkComletionEntity = new Entity();
      private Action<Undercut> finishInstallTbmEvent;
      private Action<TailTunnel> finishInstallSwitchEvent;

        public TunnelFederateControl()
      {
        InitializeComponent();
            this.eventEngine.InitializeEngine();
            this.tunnelInfoHost.EventEngine = this.eventEngine;

            this.constructTailTunnelEvent                    =                    new
Action<TailTunnel>(ConstructTailTunnel);
```

```
            this.installTbmEvent = new Action<Undercut>(InstallTbm);
            this.installSwitchEvent = new Action<TailTunnel>(InstallSwitch);
            this.checkCompletionEvent = new Action<Entity>(CheckCompletion);

        this.ShiftOffEvent = new Action<Entity>(ShiftOffEventHandler);
        this.ShiftOnEvent = new Action<Entity>(ShiftOnEventHandler);
        this.finishInstallTbmEvent = new Action<Undercut>(FinishInstallTbm);
        this.finishInstallSwitchEvent                    =                    new
Action<TailTunnel>(FinishInstallSwitch);
      }

      private       void       fedAmb_TimeAdvanceGrant(object       sender,
TimeAdvanceGrantEventArgs e)
      {
            this.eventEngine.Simulate(e.theTime);

        rtiAmb.NextMessageRequest(Math.Min(this.eventEngine.TimeNext,
this.fedAmb.NextLocalAttributeUpdateTime));
      }

      private     void     shaftFactory_ReflectAttributeValues(object     sender,
ReflectAttributeValuesEventArgs e)
      {
            if (e.theValues.Contains(shaftAttributes.State.GetAttributeHandle()))
            {
                Shaft shaft = this.shaftFactory[e.theObject];
                if (shaft.State == ShaftState.Finished)
                {
                        foreach (TailTunnel tailTunnel in this.tailTunnelFactory)
                                if (tailTunnel.WorkingShaft == e.theObject)
                                        StartSimulation(tailTunnel, e.theTime);

                        foreach (Undercut undercut in this.undercutFactory)
                if (undercut.WorkingShaft == e.theObject)
                   StartSimulation(undercut, e.theTime);
                }
            }
      }

        private void fedAmb_BeginExecution(object sender, EventArgs e)
        {
            this.eventEngine.InitializeScenario();
            this.tunnelInfoHost.BeginExecution();
            if (this.tunnelFactory.Count == 0)
                this.rtiAmb.ReadyToTerminate();
        this.allEntities.Add(this.checkComletionEntity);
        }

        private void tunnelFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
        {

        }

        private void fedAmb_EndExecution(object sender, EventArgs e)
        {
            this.eventEngine.FinalizeScenario();
        }

        #region Private TailTunnel & Undercut Methods

        #region GetExcavationTime(TailTunnelSection)
        /// <summary>
        /// Gets the time taken to excavate a tail tunnel section.
        /// </summary>
        /// <param name="section">The tail tunnel section to get the excavation
time of.</param>
        /// <returns>The excavation time of the tail tunnel section, in standard
units (seconds).</returns>
        private double GetExcavationTime(TailTunnelSection section)
        {
            double undercutVolume = GetExcavatedDirtVolume(section);
            return undercutVolume / section.Productivity.Sample();
        }
        #endregion

        #region GetExcavationTime(TailTunnelSection, double)
        /// <summary>
        /// Gets the time taken to excavate a tail tunnel section, based on the final
excavated dirt volume.
        /// </summary>
        /// <param name="section">The tail tunnel section to get the excavation
time of.</param>
        /// <param name="excavatedDirtVolume">The final volume of dirt
excavated from the tail tunnel section.</param>
        /// <returns>The excavation time of the tail tunnel section, in standard
units (seconds).</returns>
        private double GetExcavationTime(TailTunnelSection section, double
excavatedDirtVolume)
        {
            return excavatedDirtVolume / section.Productivity.Sample();
        }
        #endregion

        #region GetExcavatedDirtVolume(TailTunnelSection)
        /// <summary>
        /// Gets the final volume of dirt excavated from a tail tunnel section.
        /// </summary>
        /// <param name="section">The tail tunnel section to get the final
excavated dirt volume of.</param>
        /// <returns>The final volume of dirt excavated from the tail tunnel
section.</returns>
        private double GetExcavatedDirtVolume(TailTunnelSection section)
        {
            return section.Length * section.GetTailTunnel().ExcavationArea *
section.SoilSwellFactor;
        }
        #endregion

        #region GetExcavatedDirtVolume(TailTunnel)
        /// <summary>
        /// Gets the final volume of dirt excavated from a tail tunnel.
        /// </summary>
        /// <param name="tailTunnel">The tail tunnel to get the final excavated
dirt volume of.</param>
        /// <returns>The final volume of dirt excavated from the tail
tunnel.</returns>
        private double GetExcavatedDirtVolume(TailTunnel tailTunnel)
        {
            TailTunnelSection section = tailTunnel.GetFirstSection();
            double volume = 0.0;
            while (section != null)
            {
                volume += GetExcavatedDirtVolume(section);
                section = section.GetNextSection();
            }
            return volume;
        }
        #endregion

        #region UpdateTailTunnel<T>(TailTunnelUpdateEntity<T>)
        private void UpdateTailTunnel<T>(TailTunnelUpdateEntity<T> entity)
        {
            entity.ObjectInstance.AttributeOwnershipAcquisition(entity.Attribute);
            entity.ObjectInstance.UpdateAttributeValues(entity.Time,
entity.Attribute, entity.Value);
```

```
    entity.ObjectInstance.UnconditionalAttributeOwnershipDivestiture(entity.Attr
ibute);
        }
        #endregion

        #region
UpdateTailTunnelSection<T>(TailTunnelSectionUpdateEntity<T>)
        private                                                          void
UpdateTailTunnelSection<T>(TailTunnelSectionUpdateEntity<T> entity)
        {
            entity.ObjectInstance.AttributeOwnershipAcquisition(entity.Attribute);
            entity.ObjectInstance.UpdateAttributeValues(entity.Time,
entity.Attribute, entity.Value);

    entity.ObjectInstance.UnconditionalAttributeOwnershipDivestiture(entity.Attr
ibute);
        }
        #endregion

        #region UpdateUndercut<T>(UndercutUpdateEntity<T>)
        private void UpdateUndercut<T>(UndercutUpdateEntity<T> entity)
        {
            entity.ObjectInstance.AttributeOwnershipAcquisition(entity.Attribute);
            entity.ObjectInstance.UpdateAttributeValues(entity.Time,
entity.Attribute, entity.Value);

    entity.ObjectInstance.UnconditionalAttributeOwnershipDivestiture(entity.Attr
ibute);
        }
        #endregion

        #region StartSimulation(TailTunnel, double)
        /// <summary>
        /// Starts simulation of a tail tunnel.
        /// </summary>
        /// <param name="tailTunnel">The tail tunnel to start simulating</param>
        /// <param name="time">The logical time at which to start the next step in
the simulation</param>
        private void StartSimulation(TailTunnel tailTunnel, double theTime)
        {

    tailTunnel.AttributeOwnershipAcquisition(tailTunnelAttributes.StartTime);
            tailTunnel.UpdateAttributeValues(tailTunnelAttributes.StartTime,
theTime);
```

```
     tailTunnel.UnconditionalAttributeOwnershipDivestiture(tailTunnelAttributes.
StartTime);

                TailTunnelSection firstSection = tailTunnel.GetFirstSection();

     firstSection.AttributeOwnershipAcquisition(tailTunnelSectionAttributes.Start
Time);

     firstSection.UpdateAttributeValues(tailTunnelSectionAttributes.StartTime,
theTime);

     firstSection.UnconditionalAttributeOwnershipDivestiture(tailTunnelSectionAt
tributes.StartTime);

                if (tailTunnel.ExcavationArea > 0.0)
                {
                     // set CurrentSection to FirstSection
                     this.eventEngine.ScheduleEvent(new
TailTunnelUpdateEntity<ObjectInstanceHandle>(tailTunnel,
tailTunnelAttributes.CurrentSection,                    tailTunnel.FirstSection,
Math.Ceiling(theTime + 1.0)),
                               new
Action<TailTunnelUpdateEntity<ObjectInstanceHandle>>(UpdateTailTunnel<O
bjectInstanceHandle>), 0);

                     // set section state to Excavating
                     this.eventEngine.ScheduleEvent(new
TailTunnelSectionUpdateEntity<TailTunnelSectionState>(firstSection,
tailTunnelSectionAttributes.State,             TailTunnelSectionState.Excavating,
Math.Ceiling(theTime + 1.0)),
                               new
Action<TailTunnelSectionUpdateEntity<TailTunnelSectionState>>(UpdateTailT
unnelSection<TailTunnelSectionState>), 0);

                     this.eventEngine.ScheduleEvent(tailTunnel,
this.constructTailTunnelEvent, theTime - this.eventEngine.TimeNow);
                }
                else // the tail tunnel doesn't exist in the real world, it just exists here to
host a switch
                {

     tailTunnel.AttributeOwnershipAcquisition(tailTunnelAttributes.FinishTime);
                     tailTunnel.UpdateAttributeValues(tailTunnelAttributes.FinishTime,
theTime);
```

```
    tailTunnel.UnconditionalAttributeOwnershipDivestiture(tailTunnelAttributes.
FinishTime);


    firstSection.AttributeOwnershipAcquisition(tailTunnelSectionAttributes.Finis
hTime);

    firstSection.UpdateAttributeValues(tailTunnelSectionAttributes.FinishTime,
theTime);

    firstSection.UnconditionalAttributeOwnershipDivestiture(tailTunnelSectionAt
tributes.FinishTime);

                    // set section state to Finished
                    this.eventEngine.ScheduleEvent(new
TailTunnelSectionUpdateEntity<TailTunnelSectionState>(firstSection,
tailTunnelSectionAttributes.State,                    TailTunnelSectionState.Finished,
Math.Ceiling(theTime + 1.0)),
                            new
Action<TailTunnelSectionUpdateEntity<TailTunnelSectionState>>(UpdateTailT
unnelSection<TailTunnelSectionState>), 0);
                }
            }
            #endregion

            #region StartSimulation(Undercut, double)
            /// <summary>
            /// Starts simulation of a undercut.
            /// </summary>
            /// <param name="tailTunnel">The undercut to start simulating</param>
            /// <param name="time">The logical time at which to start the next step in
the simulation</param>
            private void StartSimulation(Undercut undercut, double time)
            {

    undercut.AttributeOwnershipAcquisition(undercutAttributes.StartTime);
                undercut.UpdateAttributeValues(undercutAttributes.StartTime, time);

    undercut.UnconditionalAttributeOwnershipDivestiture(undercutAttributes.Sta
rtTime);

                TailTunnelSection firstSection = undercut.GetFirstSection();

    firstSection.AttributeOwnershipAcquisition(tailTunnelSectionAttributes.Start
Time);
```

```
     firstSection.UpdateAttributeValues(tailTunnelSectionAttributes.StartTime, 
time);

     firstSection.UnconditionalAttributeOwnershipDivestiture(tailTunnelSectionAt
tributes.StartTime);

               if (undercut.ExcavationArea > 0.0)
               {
                    // set CurrentSection to FirstSection
                    this.eventEngine.ScheduleEvent(new 
UndercutUpdateEntity<ObjectInstanceHandle>(undercut, 
undercutAttributes.CurrentSection, undercut.FirstSection, Math.Ceiling(time + 
1.0)),
                              new 
Action<UndercutUpdateEntity<ObjectInstanceHandle>>(UpdateUndercut<Objec
tInstanceHandle>), 0);

                    // set section state to Excavating
                    this.eventEngine.ScheduleEvent(new 
TailTunnelSectionUpdateEntity<TailTunnelSectionState>(firstSection, 
tailTunnelSectionAttributes.State, TailTunnelSectionState.Excavating, 
Math.Ceiling(time + 1.0)),
                              new 
Action<TailTunnelSectionUpdateEntity<TailTunnelSectionState>>(UpdateTailT
unnelSection<TailTunnelSectionState>), 0);

                    this.eventEngine.ScheduleEvent(undercut, 
this.constructTailTunnelEvent, time - this.eventEngine.TimeNow);
               }
               else // the tail tunnel doesn't exist in the real world, it just exists here to 
host a switch
               {

     undercut.AttributeOwnershipAcquisition(undercutAttributes.FinishTime);
                    undercut.UpdateAttributeValues(undercutAttributes.FinishTime, 
time);

     undercut.UnconditionalAttributeOwnershipDivestiture(undercutAttributes.Fin
ishTime);


     firstSection.AttributeOwnershipAcquisition(tailTunnelSectionAttributes.Finis
hTime);
```

```
    firstSection.UpdateAttributeValues(tailTunnelSectionAttributes.FinishTime,
time);

    firstSection.UnconditionalAttributeOwnershipDivestiture(tailTunnelSectionAt
tributes.FinishTime);

                // set section state to Finished
                this.eventEngine.ScheduleEvent(new
TailTunnelSectionUpdateEntity<TailTunnelSectionState>(firstSection,
tailTunnelSectionAttributes.State,                TailTunnelSectionState.Finished,
Math.Ceiling(time + 1.0)),
                    new
Action<TailTunnelSectionUpdateEntity<TailTunnelSectionState>>(UpdateTailT
unnelSection<TailTunnelSectionState>), 0);

                // set CurrentInstallationProcess to UndercutInstallation.Tbm
                this.eventEngine.ScheduleEvent(new
UndercutUpdateEntity<UndercutInstallation>(undercut,
undercutAttributes.CurrentInstallationProcess,          UndercutInstallation.Tbm,
Math.Ceiling(time + 1.0)),
                    new
Action<UndercutUpdateEntity<UndercutInstallation>>(UpdateUndercut<Underc
utInstallation>), 0);

                this.eventEngine.ScheduleEvent(undercut,     this.installTbmEvent,
time - this.eventEngine.TimeNow);
            }
        }
        #endregion

        #region                    undercutFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
        private   void   undercutFactory_ReflectAttributeValues(object   sender,
ReflectAttributeValuesEventArgs e)
        {
            if
(e.theValues.Contains(undercutAttributes.CurrentInstallationProcess.GetAttribute
Handle()))
            {
                Undercut undercut = this.undercutFactory[e.theObject];
                if            (undercut.CurrentInstallationProcess             ==
UndercutInstallation.Switch)
                {
                    this.eventEngine.ScheduleEvent(undercut,
this.installSwitchEvent, e.theTime - this.eventEngine.TimeNow);
```

```
                    if (undercut.TailTunnel != ObjectInstanceHandle.Empty)

    this.eventEngine.ScheduleEvent(undercut.GetTailTunnel(),
this.installSwitchEvent, e.theTime - this.eventEngine.TimeNow);
                }
            }
        }
        #endregion

        #region Tail Tunnel Simulation Events

        #region ConstructTailTunnel(TailTunnel)
        private void ConstructTailTunnel(TailTunnel tailTunnel)
        {
            double                tailTunnelExcavatedDirtVolume              =
GetExcavatedDirtVolume(tailTunnel);
            double duration = GetExcavationTime(tailTunnel.GetFirstSection(),
tailTunnelExcavatedDirtVolume);


    tailTunnel.AttributeOwnershipAcquisition(tailTunnelAttributes.FinishTime,
tailTunnelAttributes.ExcavatedDirtVolume);
            List<ObjectAttributeValuePair<TailTunnel>> taiTunnelFinishUpdates
= new List<ObjectAttributeValuePair<TailTunnel>>(3);
            taiTunnelFinishUpdates.Add(tailTunnelAttributes.FinishTime,
this.eventEngine.TimeNow + duration);

    taiTunnelFinishUpdates.Add(tailTunnelAttributes.ExcavatedDirtVolume,
tailTunnelExcavatedDirtVolume);
            tailTunnel.UpdateAttributeValues(this.eventEngine.TimeNow          +
duration, taiTunnelFinishUpdates);

    tailTunnel.UnconditionalAttributeOwnershipDivestiture(tailTunnelAttributes.
FinishTime, tailTunnelAttributes.ExcavatedDirtVolume);

            Scenario scenario = tailTunnel.GetWorkingShaft().GetScenario();

    scenario.AttributeOwnershipAcquisition(scenarioAttributes.MaterialCost);
            scenario.UpdateAttributeValues(this.eventEngine.TimeNow          +
duration,      scenarioAttributes.MaterialCost,      scenario.MaterialCost      +
tailTunnel.GetFirstSection().MaterialCostPerMeter                             *
tailTunnel.GetCompleteLength());
            scenario.UnconditionalAttributeOwnershipDivestiture();

            if (tailTunnel.GetType() == typeof(Undercut))
            {
                    Undercut undercut = (Undercut)tailTunnel;

    undercut.AttributeOwnershipAcquisition(undercutAttributes.CurrentInstallati
onProcess);
                    undercut.UpdateAttributeValues(this.eventEngine.TimeNow        +
duration,                              undercutAttributes.CurrentInstallationProcess,
UndercutInstallation.Tbm);

    undercut.UnconditionalAttributeOwnershipDivestiture(undercutAttributes.Cur
rentInstallationProcess);

                    this.eventEngine.ScheduleEvent(undercut,      this.installTbmEvent,
duration);
                }
                else
                {
              this.eventEngine.ScheduleEvent(this.checkComletionEntity,
this.checkCompletionEvent, duration);
                }
            }
            #endregion

            #region InstallTbm(Undercut)
            private void InstallTbm(Undercut undercut)
            {
            ErrorLog.AddError("       START       install       TBM.@       "       +
this.theCalendar.calculateCurrentDateTime(fedAmb.CurrentTime));
            double                              duration                              =
Math.Max((undercut.GetTunnel().GetTbm().InstallationDuration.Sample()         -
this.fedAmb.Lookahead), this.fedAmb.Lookahead);
            this.eventEngine.ScheduleEvent(undercut,       this.finishInstallTbmEvent,
duration);
            }
            #endregion

        #region FinishInstallTbm(Undercut)
        private void FinishInstallTbm(Undercut undercut)
        {
            double duration = this.fedAmb.Lookahead;


undercut.AttributeOwnershipAcquisition(undercutAttributes.CurrentInstallationPr
ocess);
            undercut.UpdateAttributeValues(this.eventEngine.TimeNow  +  duration,
undercutAttributes.CurrentInstallationProcess, UndercutInstallation.None);
```

```
undercut.UnconditionalAttributeOwnershipDivestiture(undercutAttributes.Curren
tInstallationProcess);

          this.eventEngine.ScheduleEvent(this.checkComletionEntity,
this.checkCompletionEvent, duration);
          ErrorLog.AddError("     finish     install     tbm.@     "     +
this.theCalendar.calculateCurrentDateTime(fedAmb.CurrentTime));

          // Send an interaction declaring the completion of TBM installation.
          this.tbmInstalledTerminal.TailTunnel                              =
undercut.GetObjectInstanceHandle();

this.tbmInstalledTerminal.SendInteraction(this.rtiAmb.QueryLogicalTime()     +
this.fedAmb.Lookahead);
       }
       #endregion

          #region InstallSwitch(TailTunnel)
          private void InstallSwitch(TailTunnel tailTunnel)
          {
          ErrorLog.AddError("     START     install     switch.@     "     +
this.theCalendar.calculateCurrentDateTime(fedAmb.CurrentTime));
          double   duration   =   tailTunnel.SwitchInstallationDuration.Sample()   -
this.fedAmb.Lookahead;
          duration = Math.Max(duration, this.fedAmb.Lookahead);

          this.eventEngine.ScheduleEvent(tailTunnel,  this.finishInstallSwitchEvent,
duration);
          }
          #endregion

       #region FinishInstallSwitch(TailTunnel)
       private void FinishInstallSwitch(TailTunnel tailTunnel)
       {
          double duration = this.fedAmb.Lookahead;


tailTunnel.AttributeOwnershipAcquisition(tailTunnelAttributes.HasSwitch);
          tailTunnel.UpdateAttributeValues(this.eventEngine.TimeNow + duration,
tailTunnelAttributes.HasSwitch, true);

tailTunnel.UnconditionalAttributeOwnershipDivestiture(tailTunnelAttributes.Has
Switch);

          if (tailTunnel.GetType() == typeof(Undercut))
          {
            Undercut undercut = (Undercut)tailTunnel;

undercut.AttributeOwnershipAcquisition(undercutAttributes.CurrentInstallationPr
ocess);
            undercut.UpdateAttributeValues(this.eventEngine.TimeNow + duration,
undercutAttributes.CurrentInstallationProcess, UndercutInstallation.None);

undercut.UnconditionalAttributeOwnershipDivestiture(undercutAttributes.Curren
tInstallationProcess);

            // Send an interaction declaring the completion of switch installation.
            this.switchInstalledTerminal.Undercut                              =
undercut.GetObjectInstanceHandle();

this.switchInstalledTerminal.SendInteraction(this.rtiAmb.QueryLogicalTime()   +
duration);
          }

          this.eventEngine.ScheduleEvent(this.checkComletionEntity,
this.checkCompletionEvent, duration);
          ErrorLog.AddError("      finish      install      switch.@      "      +
this.theCalendar.calculateCurrentDateTime(fedAmb.CurrentTime));
        }
        #endregion

          #region CheckCompletion(Entity)
          /// <summary>
          /// Checks the completion of the simulation with regard to this federate,
          /// and calls rtiAmb.ReadyToTerminate() iff complete.
          /// </summary>
          /// <param name="entity">An irrelevant entity</param>
          private void CheckCompletion(Entity entity)
          {
              foreach (Undercut undercut in this.undercutFactory)
              {
                  // if the undercut is not complete, return
                  if (undercut.GetState() != TunnelSectionState.Complete)
                          return;

                  // if the undercut should eventually have a switch and does not,
return
                  if
(!double.IsNaN(undercut.GetTunnel().SwitchInstallationChainage)

    && !double.IsInfinity(undercut.GetTunnel().SwitchInstallationChainage)
```

```
                        && !(undercut.GetTunnel().SwitchInstallationChainage >
undercut.GetTunnel().GetFinishedLength())
                        && !undercut.HasSwitch)
                        return;
            }
            foreach (TailTunnel tailTunnel in this.tailTunnelFactory)
            {
                // if the undercut is not complete, return
                if (tailTunnel.GetState() != TunnelSectionState.Complete)
                        return;

                // if the undercut should eventually have a switch and does not,
return
                if
(!double.IsNaN(tailTunnel.GetTunnel().SwitchInstallationChainage)

    && !double.IsInfinity(tailTunnel.GetTunnel().SwitchInstallationChainage)
                        && !(tailTunnel.GetTunnel().SwitchInstallationChainage >
tailTunnel.GetTunnel().GetFinishedLength())
                        && !tailTunnel.HasSwitch)
                        return;
            }
            this.rtiAmb.ReadyToTerminate();
        }
        #endregion

       # region ShiftOnEventHandler
       /// <summary>
       /// try to resume all suspended entities when shift changes to ON
       /// </summary>
       /// <param name="entity"></param>
       private void ShiftOnEventHandler(Entity entity)
       {
          this.suspendedEntities.ResumeEntities(eventEngine);
          this.suspendedEntities.UnionWith(this.allEntities);
       }
       #endregion

       # region ShiftOffEventHandler
       /// <summary>
       /// try to suspend all scheduled entities when shift changes to OFF
       /// </summary>
       /// <param name="entity"></param>
       private void ShiftOffEventHandler(Entity entity)
       {
          this.suspendedEntities.SuspendEntitiesIfNeed(eventEngine);
    }
    #endregion

        #endregion

        #endregion

        #region Private Nested Classes

        #region TailTunnelSectionUpdateEntity<T>
        private class TailTunnelSectionUpdateEntity<T> : Entity
        {
            public TailTunnelSection ObjectInstance { get; private set; }
            public ObjectAttribute<TailTunnelSection, T> Attribute { get; private
set; }
            public T Value { get; private set; }
            public double Time { get; private set; }

            public TailTunnelSectionUpdateEntity(TailTunnelSection section,
ObjectAttribute<TailTunnelSection, T> attribute, T value, double time)
            {
                this.ObjectInstance = section;
                this.Attribute = attribute;
                this.Value = value;
                this.Time = time;
            }
        }
        #endregion

        #region TailTunnelUpdateEntity<T>
        private class TailTunnelUpdateEntity<T> : Entity
        {
            public TailTunnel ObjectInstance { get; private set; }
            public ObjectAttribute<TailTunnel, T> Attribute { get; private set; }
            public T Value { get; private set; }
            public double Time {get; private set;}

            public TailTunnelUpdateEntity(TailTunnel tailTunnel,
ObjectAttribute<TailTunnel, T> attribute, T value, double time)
            {
                this.ObjectInstance = tailTunnel;
                this.Attribute = attribute;
                this.Value = value;
                this.Time = time;
            }
        }
```

```
      #endregion

      #region UndercutUpdateEntity<T>
      private class UndercutUpdateEntity<T> : Entity
      {
         public Undercut ObjectInstance { get; private set; }
         public ObjectAttribute<Undercut, T> Attribute { get; private set; }
         public T Value { get; private set; }
         public double Time {get; private set;}

         public UndercutUpdateEntity(Undercut undercut,
ObjectAttribute<Undercut, T> attribute, T value, double time)
         {
            this.ObjectInstance = undercut;
            this.Attribute = attribute;
            this.Value = value;
            this.Time = time;
         }
      }
      #endregion

    private void undercutFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
    {
      this.allEntities.Add(undercutFactory[e.theObject]);
    }

      #endregion

    private void myShiftFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
    {
      this.theShift = myShiftFactory[e.theObject];
    }

    private void myShiftFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
    {
      var shift = myShiftFactory[e.theObject];
      if (e.theValues.Contains(shiftAttributes.IsWorking.GetAttributeHandle()))
      {
        if (e.theTime > 0)
        {
          double timeOfUpdate = Math.Max(e.theTime,
this.rtiAmb.QueryLogicalTime() + fedAmb.Lookahead);
          if (shift.IsWorking)
```

```
                {
                    eventEngine.ScheduleEvent(this.shiftControlEntity, ShiftOnEvent,
e.theTime - eventEngine.TimeNow);
                }
                else
                {
                    eventEngine.ScheduleEvent(this.shiftControlEntity, ShiftOffEvent,
e.theTime - eventEngine.TimeNow);
                }
            }
        }
    }

    private void tailTunnelFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
    {
        this.allEntities.Add(tailTunnelFactory[e.theObject]);
    }

    private void calendarFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
    {
        this.theCalendar = calendarFactory[e.theObject];
    }
  }
}
```

c. Tunnel Federate II

```
using System;
using System.Collections;
using System.Collections.Generic;

using Cosye.Framework;
using Cosye.Hla.Rti;
using Simphony;
using Simphony.Mathematics;
using Simphony.Simulation;

namespace Cosye.Tunneling.TunnelFederate
{

    public partial class TunnelFederate : FederateControl
    {
        #region Private Readonly Fields

      private readonly List<string> theShaftAttributes = new List<string>();
      private readonly List<string> theUndercutAttributes = new List<string>();

      private readonly List<string> theTunnelAttributes = new List<string>();
      private readonly List<string> theGeotechnicalSectionAttributes = new
List<string>();

      private readonly List<string> theTrainAttributes1 = new List<string>();
      private readonly List<string> theTbmAttributes = new List<string>();

      private readonly Dictionary<ObjectInstanceHandle, Resource> myTBMs =
new Dictionary<ObjectInstanceHandle, Resource>();
      private readonly Dictionary<ObjectInstanceHandle, WaitingFile>
myTbmFiles = new Dictionary<ObjectInstanceHandle, WaitingFile>();
      private readonly Dictionary<ObjectInstanceHandle, bool>
delayedRequestTbm = new Dictionary<ObjectInstanceHandle, bool>();

      // this Dictionary is used where Tbm.WorkState used to be used,
      // because the WorkState updates have to be timed in order for the GUI to
work,
      // but just changing the update causes the code to break.
      private readonly Dictionary<Tbm, TbmWorkState> WorkState = new
Dictionary<Tbm, TbmWorkState>();

      # endregion
```

```
   #region Private Fields

 private Entity breakdownEntity = new Entity();
 private Entity shiftControlEntity = new Entity();
 private Calendar theCalendar;
 private DateTime startDateTime = DateTime.Today;
 private Action<Entity> ShiftOnEvent;
 private Action<Entity> ShiftOffEvent;
 private Action<Tunnel> FinalizeOneMeterEvent;
 private ShiftAwareEntities allEntities = new ShiftAwareEntities();
 private ShiftAwareEntities suspendedEntities = new ShiftAwareEntities();
 private Shift theShift;
 private Tunnel nestTunnel;
 private bool firstShiftControl=true;
 private double lastChainage = 0;

 private Action<BreakdownEntity> BreakdownEvent;
 private Action<BreakdownEntity> RepairEvent;
 private Action<Entity> EmptyActionEvent;
 private Action<Tunnel> ProcessWaitingTrainEvent;
 private ConcreteLinerInventory myConcrete;

 private      BasicDiscreteEventEngine      MyEngine      =      new
BasicDiscreteEventEngine();

 private Action<Tunnel> StartOneGeotechnicalSectionEvent;
 private Action<Tunnel> ExcavatingEvent;
 private Action<Tunnel> ResettingEvent;
 private Action<Tunnel> LiningEvent;

 private Action<Tunnel> ExtendUtilityEvent;
 private Action<Tunnel> ExtendGantryConveyerEvent;
 private Action<Tunnel> ExtendTrackEvent;
 private Action<Tunnel> SurveyEvent;

 private Action<Train> ReleaseTbmEvent;
 private Action<Train> TbmCapturedEvent;
 private Action<Train> RequestTbmEvent;

 #endregion

   public TunnelFederate()
   {
      InitializeComponent();

      theShaftAttributes.Add(shaftAttributes.State);
```

```
            theShaftAttributes.Add(shaftAttributes.Name);

            theUndercutAttributes.Add(undercutAttributes.StartTime);
            theUndercutAttributes.Add(undercutAttributes.FinishTime);
            theUndercutAttributes.Add(undercutAttributes.ExcavatedDirtVolume);

            theTunnelAttributes.Add(tunnelAttributes.ExcavationArea);
            theTunnelAttributes.Add(tunnelAttributes.FinishTime);
            theTunnelAttributes.Add(tunnelAttributes.ExcavatedDirtVolume);

    theTunnelAttributes.Add(tunnelAttributes.CurrentGeotechnicalSection);
            theTunnelAttributes.Add(tunnelAttributes.CurrentChainage);
            theTunnelAttributes.Add(tunnelAttributes.State);
            theTunnelAttributes.Add(tunnelAttributes.StartTime);
            theTunnelAttributes.Add(tunnelAttributes.CrewActivity);


    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.LinerTy
pe);

    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.StartTi
me);

    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.FinishTi
me);

    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.State);

            theTrainAttributes1.Add(trainAttributes.CurrentDirtVolume);
            theTrainAttributes1.Add(trainAttributes.CartsCounter);
            theTrainAttributes1.Add(trainAttributes.EnergyConsumption);
            theTrainAttributes1.Add(trainAttributes.Position);
            theTrainAttributes1.Add(trainAttributes.WorkState);

            theTbmAttributes.Add(tbmAttributes.ResourceState);
            theTbmAttributes.Add(tbmAttributes.EmissionFactor);
            theTbmAttributes.Add(tbmAttributes.EnergyConsumption);
            theTbmAttributes.Add(tbmAttributes.WorkState);
            theTbmAttributes.Add(tbmAttributes.LiningDuration);
            theTbmAttributes.Add(tbmAttributes.PenetrationRate);
            theTbmAttributes.Add(tbmAttributes.WorkHours);
            theTbmAttributes.Add(tbmAttributes.ResettingDuration);
            // theTbmAttributes.Add(tbmAttributes.CostPerHour);
            theTbmAttributes.Add(tbmAttributes.WorkHours);

            // Initialize events.
            BreakdownEvent                    =                    new
Action<BreakdownEntity>(BreakdownEventHandler);
            RepairEvent = new Action<BreakdownEntity>(RepairEventHandler);
            EmptyActionEvent = new Action<Entity>(EmptyAction);
            ShiftOffEvent = new Action<Entity>(ShiftOffEventHandler);
            ShiftOnEvent = new Action<Entity>(ShiftOnEventHandler);
            ProcessWaitingTrainEvent                =                new
Action<Tunnel>(ProcessWaitingTrain);
        FinalizeOneMeterEvent = new Action<Tunnel>(FinalizeOneMeter);

            ExcavatingEvent = new Action<Tunnel>(Excavating);
            ResettingEvent = new Action<Tunnel>(Resetting);
            LiningEvent = new Action<Tunnel>(Lining);

            StartOneGeotechnicalSectionEvent          =          new
Action<Tunnel>(StartOneGeotechnicalSection);

            ExtendUtilityEvent = new Action<Tunnel>(ExtendUtility);
            ExtendGantryConveyerEvent             =             new
Action<Tunnel>(ExtendGantryConveyer);
            ExtendTrackEvent = new Action<Tunnel>(ExtendTrack);
            SurveyEvent = new Action<Tunnel>(Survey);

            RequestTbmEvent = new Action<Train>(RequestTbm);
            TbmCapturedEvent = new Action<Train>(TbmCaptured);
            ReleaseTbmEvent = new Action<Train>(ReleaseTbm);

            MyEngine.InitializeEngine();
            this.tunnelInfoHost.EventEngine = this.MyEngine;
        }

        #region Private Methods

      private Train getTheTrainAtTunnelFace(Tunnel theTunnel)
      {
        Train currentTrain = null;
        foreach (Train theTrain in MyTrainFactory)
        {
          if   (theTrain.Tunnel   ==   theTunnel.GetObjectInstanceHandle()   &&
theTrain.Position == TrainLocation.TunnelFace)
          {
            currentTrain = theTrain;
          }
        }
        return currentTrain;
      }
```

```
        private Train getTheTrainOwningTbm(Tunnel theTunnel)
        {
            Train currentTrain = null;
            var resource = myTBMs[theTunnel.GetObjectInstanceHandle()];
            foreach (Train theTrain in MyTrainFactory)
            {
                if (resource.ServersOwnedByEntity(theTrain) > 0)
                {
                    currentTrain = theTrain;
                }
            }
            return currentTrain;
        }

        #region calculateDateTime(double)
        private DateTime calculateDateTime(double logicTime)
        {
            if (logicTime > 0 && !double.IsInfinity(logicTime))
                return this.theCalendar.StartDateTime.AddSeconds(logicTime);
            else return DateTime.MaxValue;
        }
        #endregion

        private void CheckUtility(Tunnel theTunnel, double duration)
        {
            ErrorLog.AddError("CheckUtility          entered          @          "          +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            Tbm theTbm = theTunnel.GetTbm();
            WorkState[theTbm] = TbmWorkState.Waiting;
            theTbm.UpdateAttributeValues(MyEngine.TimeNow         +         duration,
tbmAttributes.WorkState, TbmWorkState.Waiting);

            if (theTunnel.ExtendUtilityInterval == 0)
                CheckGantry(theTunnel, duration);
            else
            {
                double          Check          =          theTunnel.CurrentChainage          %
theTunnel.ExtendUtilityInterval;
                if (Check == 0)
                {
                    ErrorLog.AddError("CheckUtility conditions passed, scheduled @ "
+ this.calculateDateTime(MyEngine.TimeNow + duration).ToString());
                    MyEngine.ScheduleEvent(theTunnel, ExtendUtilityEvent, duration);
                    theTunnel.UpdateAttributeValues(MyEngine.TimeNow  +  duration,
tunnelAttributes.CrewActivity, TunnelCrewActivity.ExtendingUtility);
```

```
            }
            else CheckGantry(theTunnel, duration);
          }
        }

        private void CheckGantry(Tunnel theTunnel, double duration)
        {
          ErrorLog.AddError("CheckGantry        entered              @        "        +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
          double Check = 1;
          if (theTunnel.CurrentChainage <= theTunnel.GetTbm().GantryLength)
          {
            Check              =              theTunnel.CurrentChainage              %
theTunnel.GetTbm().GantrySectionLength;
          }
          else if (theTunnel.CurrentChainage == theTunnel.GetTbm().GantryLength
+ theTunnel.GetTbm().GantrySectionLength)
          { Check = 0; }

          if (Check == 0)
          {
            ErrorLog.AddError("CheckGantry conditions passed, scheduled @ " +
this.calculateDateTime(MyEngine.TimeNow + duration).ToString());
            MyEngine.ScheduleEvent(theTunnel,      ExtendGantryConveyerEvent,
duration);
            theTunnel.UpdateAttributeValues(MyEngine.TimeNow    +    duration,
tunnelAttributes.CrewActivity, TunnelCrewActivity.ExtendingGantryConveyor);
          }
          else CheckTrack(theTunnel, duration);
        }

        private void CheckTrack(Tunnel theTunnel, double duration)
        {
          ErrorLog.AddError("CheckTrack entered");
          double        Check        =        theTunnel.CurrentChainage        %
theTunnel.ExtendTrackInterval;
          if (Check == 0)
          {
            ErrorLog.AddError("CheckTrack conditions passed, scheduled @ " +
this.calculateDateTime(MyEngine.TimeNow + duration).ToString());
            theTunnel.UpdateAttributeValues(MyEngine.TimeNow              +
duration,tunnelAttributes.CrewActivity, TunnelCrewActivity.ExtendingTrack);
            MyEngine.ScheduleEvent(theTunnel, ExtendTrackEvent, duration);
          }
          else CheckSurvey(theTunnel, duration);
        }
```

```
    private void CheckSurvey(Tunnel theTunnel, double duration)
    {
      ErrorLog.AddError("CheckSurvey entered");
      double Check = theTunnel.CurrentChainage % theTunnel.SurveyInterval;
      if (Check == 0)
      {
        ErrorLog.AddError("CheckSurvey conditions passed, schedule @ " +
this.calculateDateTime(MyEngine.TimeNow + duration).ToString());
        theTunnel.UpdateAttributeValues(MyEngine.TimeNow + duration,
tunnelAttributes.CrewActivity, TunnelCrewActivity.Survey);
        MyEngine.ScheduleEvent(theTunnel, SurveyEvent, duration);
      }
      else
      {
        ErrorLog.AddError("***++++++Schedule FinalizeOneMeterEvent @ "
          + this.calculateDateTime(Math.Max(0, duration -
fedAmb.Lookahead)).ToString());
        this.MyEngine.ScheduleEvent(theTunnel, FinalizeOneMeterEvent,
Math.Max(0, duration - fedAmb.Lookahead));
      }
    }

    private void CheckShift(Tunnel theTunnel, double duration)
    {
      if (!this.theShift.IsWorking)
      {
        ErrorLog.AddError("CheckShift entered and shift is NOT working @ "
+ calculateDateTime(rtiAmb.QueryLogicalTime()).ToString());
        TimeSpan time =
this.theCalendar.StartDateTime.AddSeconds(this.rtiAmb.QueryLogicalTime()).Ti
meOfDay;
        TimeSpan target = TimeSpan.FromSeconds(this.theShift.StartTime) +
TimeSpan.FromSeconds(this.theShift.Duration);
        double timeOfUpdate = duration + this.rtiAmb.QueryLogicalTime();
        if (time >= target) // if this is not the end of a shift
        {
          foreach (Tbm theTbm in MyTbmFactory)
          {
            theTbm.UpdateAttributeValues(timeOfUpdate,
tbmAttributes.ResourceState, ResourceState.Idle);
          }
          ErrorLog.AddError("+++ TunnelFederate.Shift Off Event when
finalizing a meter @ " +
            this.calculateDateTime(timeOfUpdate).ToString() + " Resource
Idle @ " + this.MyEngine.TimeNow.ToString());
```

```
                ErrorLog.shiftReport("+++     TunnelFederate.ShiftOffEvent     when
finalizing a meter @ " +
                   this.calculateDateTime(timeOfUpdate).ToString()  +  "  Resource
Idle @ " + this.MyEngine.TimeNow.ToString());

                MyEngine.ScheduleEvent(this.shiftControlEntity,       ShiftOffEvent,
duration);

                // Send an interaction for finalizing a tunnel meter.
                if (time >= target) // if this is not the end of a shift
                {
                   ErrorLog.AddError("send an interaction of finishing lining a meter
= " + this.calculateDateTime(timeOfUpdate).ToString());
                   ErrorLog.shiftReport("send  an  interaction  of  finishing  lining  a
meter = " + this.calculateDateTime(timeOfUpdate).ToString());
                   this.ShiftOffTerminal.Tunnel                                        =
theTunnel.GetObjectInstanceHandle();
                   this.ShiftOffTerminal.SharpOff = true;
                   this.ShiftOffTerminal.SendInteraction(timeOfUpdate);
                }
             }
          }
       }

       private   double   EnergyRateOfTbm(Tunnel   theTunnel,   TbmWorkState
currentTbmWorkState)
       {
          double Rate = 0;
          //  TBM RM100SE series 19200 which is used for NEST tunnel
          //- Excavating :    energy (KWh) = excavating time ( seconds)/3600 * 335
(KW) * 0.8
          //- Lining :  energy (KWh) = lining time ( seconds)/3600 * 100 (KW)
          //- Resetting :  energy (KWh) = lining time ( seconds)/3600 * 100 (KW)
          //- Idling : energy (KWh) = idling time ( seconds)/3600 * 10 (KW)
          GeotechnicalSection                         theSection                         =
MyGeotechnicalSectionFactory[theTunnel.CurrentGeotechnicalSection];

          if (currentTbmWorkState == TbmWorkState.Excavating)
          {
             //Electricity consumption rate of TBM excavation :
             //13,800/3600 * (-1.0816 * pr (m/h) +12.06) * operation time (second)
             double pr = theSection.NextAdvanceRate.Sample() * 3600; // what's the
unit of theSection.nextAdvanceRate? it's shown around 0.6 anytime. is it correct
value?
             Rate = 13800 / 1000 * (-1.0816 * pr + 12.06) / 3600; // (KW)
          }
          else if (currentTbmWorkState == TbmWorkState.Lining)
          {
             //Rate = 100 * 0.6; //(KW)
          }

          return Rate;
       }

       // Not used EnergyConsumptionRateOfTrain
       private double EnergyConsumptionRateOfTrain(Train theTrain)
       {

          double Rate = 0;
          // average dirt density = 120 lb/cubic feet =   1.922 (t/cubic meter)
          // the weight of one locomotive = 5897 kilograms = 5.897 t; the weight of
one empty muck car = 1 t
          // 1 m = 3.28 feet
          // assume 1t for the liner weight.
          Rate = 1.704 * (1 / theTrain.MuckCartCount + 5.897 + 1 *
theTrain.MuckCartCount) * 3.28 / 100 / 1000; // KWh /m

          return Rate;
       }

       #endregion

          #region Private Events
       private void StartOneGeotechnicalSection(Tunnel theTunnel)
       {
          ErrorLog.AddError("StartOneGeotechnicalSection event entered  @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
          Tbm theTbm = theTunnel.GetTbm();
          var prevSection = theTunnel.GetCurrentGeotechnicalSection();
          // if this is the last section, then return:
          if ((prevSection != null) && (prevSection.GetSectionNumber() ==
theTunnel.GetGeotechnicalSectionCount()))
          {
             System.Windows.Forms.MessageBox.Show("Project Completed.");
             ErrorLog.AddError("StartOneGeotechnicalSection event first condition
entered.  @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
             rtiAmb.ReadyToTerminate();
             return;
          }

          // Check if this is the 1st section:
```

```
        if (prevSection == null) // theTunnel.State == TunnelState.WaitForInstallingTBM)
        {
           ErrorLog.AddError("StartOneGeotechnicalSection event third condition entered.  @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
           theTunnel.StartTime = MyEngine.TimeNow;
           theTunnel.State = TunnelState.Startup;
           theTunnel.CurrentGeotechnicalSection = theTunnel.FirstGeotechnicalSection;
           ErrorLog.AddError("set tunnel to startup @ " + (fedAmb.CurrentTime / 60).ToString("000.0"));

           WorkState[theTbm] = TbmWorkState.Waiting;
           theTbm.UpdateAttributeValues(MyEngine.TimeNow + fedAmb.Lookahead, tbmAttributes.WorkState, TbmWorkState.Waiting);
        }
        else  // ELSE: set the next section to start
        {
           ErrorLog.AddError("StartOneGeotechnicalSection event fourth condition entered. @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());


           if (theTunnel.State == TunnelState.WaitForInstallingSwitch)
           {
              // If switch has been installed:
              ErrorLog.AddError("StartOneGeotechnicalSection event fifth condition entered. @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
              theTunnel.State = TunnelState.MultipleTrainConstructing;

              WorkState[theTbm] = TbmWorkState.Waiting;
              theTbm.UpdateAttributeValues(MyEngine.TimeNow + fedAmb.Lookahead, tbmAttributes.WorkState, TbmWorkState.Waiting);
           }
           else if ((theTunnel.State == TunnelState.Startup || theTunnel.State == TunnelState.SingleTrainConstructing))
           {
              // ELSE: when 1st section is finished, but switch is NOT installed:
              ErrorLog.AddError("StartOneGeotechnicalSection event sixth condition entered. @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());

              Undercut theUndercut = theTunnel.GetUndercut();
```

```
theUndercut.AttributeOwnershipAcquisition(undercutAttributes.CurrentInstallatio
nProcess);
            theUndercut.CurrentInstallationProcess                              =
UndercutInstallation.Switch;
            theUndercut.UpdateAttributeValues(MyEngine.TimeNow               +
fedAmb.Lookahead);

theUndercut.UnconditionalAttributeOwnershipDivestiture(undercutAttributes.Cur
rentInstallationProcess);

            theTunnel.UpdateAttributeValues(         MyEngine.TimeNow          +
fedAmb.Lookahead,tunnelAttributes.State,  TunnelState.WaitForInstallingSwitch);

            return; // must exit this function, to NOT start the next section
          }
          theTunnel.CurrentGeotechnicalSection                                  =
theTunnel.GetCurrentGeotechnicalSection().NextSection;
        }

        // initialize a new section.
        GeotechnicalSection                  theNewSection                      =
theTunnel.GetCurrentGeotechnicalSection();
        theNewSection.State = TunnelSectionState.Constructing;
        theNewSection.StartTime = MyEngine.TimeNow;

        // check if a RequestTbm event has been delayed, then resume it.
        Train theTrain = getTheTrainAtTunnelFace(theTunnel);
        if           ((theTrain           !=            null)           &&
delayedRequestTbm[theTrain.GetObjectInstanceHandle()] == true)
        {
          MyEngine.ScheduleEvent(theTrain, RequestTbmEvent, 0);
          ErrorLog.AddError("++++ RESUME delayed RequestTbm by Train:
RequestTbmEvent_ from StartOneGeotechnicalSection");
          delayedRequestTbm[theTrain.GetObjectInstanceHandle()] = false;
        }

        theTbm.UpdateAttributeValues(MyEngine.TimeNow                          +
fedAmb.Lookahead, tbmAttributes.ResourceState, ResourceState.Busy);
        theTunnel.UpdateAttributeValues(MyEngine.TimeNow                       +
fedAmb.Lookahead);  //theTunnelAttributes,
        theNewSection.UpdateAttributeValues(MyEngine.TimeNow                   +
fedAmb.Lookahead);  //theTunnelSectionAttributes,
        //System.Diagnostics.Debug.WriteLine("should processWaitTrain.");
      }
```

```
    private void RequestTbm(Train theTrain)
    {
      ErrorLog.AddError("Tunnel.RequestTbm event entered BY Train" +
theTrain.ID.ToString() + " @ " + fedAmb.CurrentTime.ToString() + "  @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
      var resource = myTBMs[theTrain.Tunnel];
      var myFile = myTbmFiles[theTrain.Tunnel];
      Tunnel theTunnel = theTrain.GetTunnel();

      if (!theTrain.IsScheduled)
      {
        this.MyEngine.RequestResource(theTrain,        resource,        1,
TbmCapturedEvent, myFile);
      }
      else
      {
        ErrorLog.AddError("*************error message: Try to CAPTURE
TBM using a train scheduled before: SKIPPED THIS TIME ************");
      }
      ErrorLog.TraceResource("TBM    REQUESTED    BY    theTrain    ;
theTrain.IsScheduled   :   "   +   theTrain.IsScheduled   +   "   @"   +
this.rtiAmb.QueryLogicalTime());
      ErrorLog.TraceResource("tbm.Available        =        "        +
resource.Available.ToString());
      ErrorLog.TraceResource("resource.InUse        =        "        +
resource.InUse.ToString());
      ErrorLog.TraceResource("resource.ServersPreempted=        "        +
resource.ServersPreempted.ToString());
      ErrorLog.TraceResource("----------------");
    }

    private void TbmCaptured(Train theTrain)
    {
      ErrorLog.AddError("TUnnel.TbmCaptured  event  entered   @  " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());

      Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];

      ErrorLog.TraceResource("TbmCaptured event: TBM CAPTURED BY
Train");
      var resource = myTBMs[theTrain.Tunnel];
      ErrorLog.TraceResource("tbm.Available        =        "        +
resource.Available.ToString());
      ErrorLog.TraceResource("resource.ServersPreempted=        "        +
resource.ServersPreempted.ToString());
      ErrorLog.TraceResource("-------------------------------");
```

```
         Tbm theTbm = theTunnel.GetTbm();
         WorkState[theTbm] = TbmWorkState.Excavating;
         theTbm.UpdateAttributeValues(this.fedAmb.CurrentTime                +
fedAmb.Lookahead, tbmAttributes.WorkState, TbmWorkState.Excavating);

         theTrain.WorkState = TrainWorkState.Loading;
         theTunnel.CrewActivity = TunnelCrewActivity.TbmOperation;

         if (theTbm.EquipmentState == EquipmentState.Functional)
         {
           ErrorLog.AddError("TbmCaptured  event  condition  passed   @  " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
           MyEngine.ScheduleEvent(theTunnel, ExcavatingEvent, 0);
           // update the current chainage of tunnel.
           if  (theTrain.CartsCounter  ==  0  |  theTunnel.CurrentChainage  >
theTbm.GantryLength)
              theTunnel.CurrentChainage += 1;
         }
         else      System.Diagnostics.Debug.WriteLine("TBM  captured  but  not
Functional.");

         theTrain.UpdateAttributeValues(this.fedAmb.CurrentTime              +
fedAmb.Lookahead);
         theTunnel.UpdateAttributeValues(this.fedAmb.CurrentTime             +
fedAmb.Lookahead);
      }

      private void Excavating(Tunnel theTunnel)
      {
         ErrorLog.AddError("Excavating     event     entered.     @     "     +
this.calculateDateTime(fedAmb.CurrentTime).ToString()
           + " Chainage = " + theTunnel.CurrentChainage.ToString()+ "-------------
---------------");
         Train theTrain = getTheTrainAtTunnelFace(theTunnel);
         if (theTrain == null)
         {
           ErrorLog.AddError("null currentTrain in excavating event.");
           System.Diagnostics.Debug.WriteLine("Null      getCurrentTrain      in
Excavating event");
           return;
         }

         GeotechnicalSection                theGeotechnicalSection               =
MyGeotechnicalSectionFactory[theTunnel.CurrentGeotechnicalSection];
         double duration;
```

```
        //dirt amount should be constrained by cart capacity and Tbm stroke length.
        double    dirtAmount    =    Math.Min((theTrain.CapacityPerCart    *
theTrain.MuckCartCount),
          (theTunnel.ExcavationArea        *        1        *
theGeotechnicalSection.SoilSwellFactor));
        Tbm theTbm = theTunnel.GetTbm();
        List<ObjectAttributeValuePair<Tbm>>    tbmUpdates    =    new
List<ObjectAttributeValuePair<Tbm>>(2);

        // if the gantry is not build successfully, then there is only one cart used for
removing dirt
        if (theTunnel.CurrentChainage <= theTbm.GantryLength)
        {
          theTrain.CartsCounter += 1;

          double count = Math.Ceiling(dirtAmount / theTrain.CapacityPerCart);
          duration = 1 / theGeotechnicalSection.NextAdvanceRate.Sample() /
count;  // the unit of nextAdvanceRate is m/second

          if (count > theTrain.CartsCounter)  // if this is NOT the last cart to load,
then ...
          {
            ErrorLog.AddError("Excavating event second condition entered. @ "
+ this.calculateDateTime(fedAmb.CurrentTime).ToString());
            theTunnel.ExcavatedDirtVolume += theTrain.CapacityPerCart;

            WorkState[theTbm] = TbmWorkState.Waiting;
            tbmUpdates.Add(tbmAttributes.WorkState, TbmWorkState.Waiting);
            theTunnel.UpdateAttributeValues(MyEngine.TimeNow + duration,
tunnelAttributes.CrewActivity, TunnelCrewActivity.WaitingTrain);
          }
          else // if this is the last cart to load, then ...
          {
            ErrorLog.AddError("Excavating event third condition entered. @ "
              + this.calculateDateTime(fedAmb.CurrentTime).ToString());

            // always fill a full cart during gantry installation phase;
            theTunnel.ExcavatedDirtVolume += theTrain.CapacityPerCart;
            theTrain.CurrentDirtVolume = theTrain.CapacityPerCart;
            theTrain.CartsCounter = 0;
            WorkState[theTbm] = TbmWorkState.Resetting;
            tbmUpdates.Add(tbmAttributes.WorkState,
TbmWorkState.Resetting);

            MyEngine.ScheduleEvent(theTunnel, ResettingEvent, duration);
```

```
                //Added by Ramzi Labban to send update on qty_on_hand to
procurement
                //** Begin **

myConcrete.AttributeOwnershipAcquisition(concreteAttributes.Quantity);
                myConcrete.Quantity--;
                myConcrete.UpdateAttributeValues(MyEngine.TimeNow              +
fedAmb.Lookahead);

myConcrete.UnconditionalAttributeOwnershipDivestiture(concreteAttributes.Qua
ntity);
                //** End**
            }
        }
        else // if the gantry has been built successfully, then a train contains many
carts in removing dirt.
        {
            ErrorLog.AddError("Excavating event fourth condition entered. @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            duration = 1 / theGeotechnicalSection.NextAdvanceRate.Sample();     //
the unit of nextAdvanceRate is m/s
            theTunnel.ExcavatedDirtVolume += dirtAmount;

            this.MyEngine.ScheduleEvent(theTunnel, ResettingEvent, duration);

            WorkState[theTbm] = TbmWorkState.Resetting;
            tbmUpdates.Add(tbmAttributes.WorkState, TbmWorkState.Resetting);
            theTrain.CurrentDirtVolume = dirtAmount;

            //Added by Ramzi Labban to send update on qty_on_hand to
procurement
            //** Begin **

myConcrete.AttributeOwnershipAcquisition(concreteAttributes.Quantity);
            myConcrete.Quantity--;
            myConcrete.UpdateAttributeValues(MyEngine.TimeNow + 61);

myConcrete.UnconditionalAttributeOwnershipDivestiture(concreteAttributes.Qua
ntity);
            //** End**
        }

        //always release tbm when finish loading
        ErrorLog.AddError(" schedule ReleaseTbmEvent from excavating event");
        MyEngine.ScheduleEvent(theTrain, ReleaseTbmEvent, duration);
```

```
            if (duration < fedAmb.Lookahead) duration = fedAmb.Lookahead;

            theTrain.UpdateAttributeValues(MyEngine.TimeNow + duration);
            theTunnel.UpdateAttributeValues(MyEngine.TimeNow + duration);
            theGeotechnicalSection.UpdateAttributeValues("SectionState",
MyEngine.TimeNow + duration);

            theTbm.EnergyConsumption       +=       EnergyRateOfTbm(theTunnel,
TbmWorkState.Excavating) * duration;//KWh
            theTbm.UpdateAttributeValues(MyEngine.TimeNow      +      duration,
tbmUpdates);
        }

        private void Resetting(Tunnel theTunnel)
        {
            ErrorLog.AddError("Resetting      event      entered      @      "      +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            double duration = theTunnel.GetTbm().ResettingDuration.Sample();
            if (duration < fedAmb.Lookahead)
               duration = fedAmb.Lookahead;

            this.MyEngine.ScheduleEvent(theTunnel,   LiningEvent,   duration);   //
preemption takes care of breakdown

            Tbm theTbm = theTunnel.GetTbm();
            theTbm.EnergyConsumption       +=       EnergyRateOfTbm(theTunnel,
TbmWorkState.Resetting) * duration / 3600;
            WorkState[theTbm] = TbmWorkState.Lining;
            List<ObjectAttributeValuePair<Tbm>>       tbmUpdates       =       new
List<ObjectAttributeValuePair<Tbm>>(2);
            tbmUpdates.Add(tbmAttributes.EnergyConsumption,
theTbm.EnergyConsumption);
            tbmUpdates.Add(tbmAttributes.WorkState, TbmWorkState.Lining);
            theTbm.UpdateAttributeValues(MyEngine.TimeNow      +      duration,
tbmUpdates);
            theTunnel.UpdateAttributeValues(MyEngine.TimeNow + duration);
        }

        private void Lining(Tunnel theTunnel)
        {
            ErrorLog.AddError("Lining      event      entered.      @      "      +
this.calculateDateTime(fedAmb.CurrentTime).ToString());

            double                    duration                    =
Math.Max(theTunnel.GetTbm().LiningDuration.Sample(), fedAmb.Lookahead);
```

```
            Tbm theTbm = theTunnel.GetTbm();
            theTbm.EnergyConsumption += EnergyRateOfTbm(theTunnel,
TbmWorkState.Lining) * duration;
            theTbm.UpdateAttributeValues(MyEngine.TimeNow + duration,
tbmAttributes.EnergyConsumption, theTbm.EnergyConsumption);

            CheckUtility(theTunnel, duration);
            CheckShift(theTunnel, duration);
        }

        private void ExtendUtility(Tunnel theTunnel)
        {
            ErrorLog.AddError("ExtendUtility event entered @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            double duration = theTunnel.ExtendUtilityDuration.Sample();

            CheckGantry(theTunnel, duration);

            Tbm theTbm = theTunnel.GetTbm();
            theTbm.EnergyConsumption += EnergyRateOfTbm(theTunnel,
TbmWorkState.Waiting) * duration;
            theTbm.UpdateAttributeValues(MyEngine.TimeNow +
fedAmb.Lookahead, tbmAttributes.EnergyConsumption,
theTbm.EnergyConsumption);
        }

        private void ExtendGantryConveyer(Tunnel theTunnel)
        {
            ErrorLog.AddError("ExtendGantryConveyer event entered @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            double duration = 0;
            if (theTunnel.GetTbm().GantryLength >= theTunnel.CurrentChainage)
            {
                duration = theTunnel.GetTbm().GantryInstallationDuration.Sample();
            }
            else if ((theTunnel.CurrentChainage - theTunnel.GetTbm().GantryLength)
== theTunnel.GetTbm().GantrySectionLength)
            {
                duration = theTunnel.GetTbm().ConveyorInstallationDuration.Sample();
                theTunnel.State = TunnelState.SingleTrainConstructing;
            }

            Tbm theTbm = theTunnel.GetTbm();
            theTbm.EnergyConsumption += EnergyRateOfTbm(theTunnel,
TbmWorkState.Waiting) * duration;
```

```
            theTbm.UpdateAttributeValues(MyEngine.TimeNow + duration,
tbmAttributes.EnergyConsumption, theTbm.EnergyConsumption);

            CheckTrack(theTunnel, duration);
            theTunnel.UpdateAttributeValues(MyEngine.TimeNow + duration);
        }

        private void ExtendTrack(Tunnel theTunnel)
        {
            ErrorLog.AddError("ExtendTrack event entered @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            double duration = theTunnel.ExtendTrackDuration.Sample();

            Tbm theTbm = theTunnel.GetTbm();
            theTbm.EnergyConsumption += EnergyRateOfTbm(theTunnel,
TbmWorkState.Waiting) * duration;
            theTbm.UpdateAttributeValues(MyEngine.TimeNow + duration,
tbmAttributes.EnergyConsumption, theTbm.EnergyConsumption);

            CheckSurvey(theTunnel, duration);
        }

        private void Survey(Tunnel theTunnel)
        {
            ErrorLog.AddError("Survey event entered @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            double duration = theTunnel.SurveyDuration.Sample();

            Tbm theTbm = theTunnel.GetTbm();
            theTbm.EnergyConsumption += EnergyRateOfTbm(theTunnel,
TbmWorkState.Waiting) * duration;
            theTbm.UpdateAttributeValues(MyEngine.TimeNow + duration,
tbmAttributes.EnergyConsumption, theTbm.EnergyConsumption);

            ErrorLog.AddError("***++++++Schedule FinalizeOneMeterEvent @ " +
this.calculateDateTime(Math.Max(0, duration - fedAmb.Lookahead)).ToString());
            this.MyEngine.ScheduleEvent(theTunnel, FinalizeOneMeterEvent,
Math.Max(0, duration - fedAmb.Lookahead));
        }

        private void FinalizeOneMeter(Tunnel theTunnel)
        {
            ErrorLog.AddError("Tunnel FinalizeOneMeter entered @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            double duration = fedAmb.Lookahead;
            double timeOfUpdate = MyEngine.TimeNow + duration;
        theTunnel.UpdateAttributeValues(timeOfUpdate,
tunnelAttributes.CrewActivity, TunnelCrewActivity.WaitingTrain);
        ErrorLog.AddError("    change    theTunnel.CrewActivity    =    "    +
theTunnel.CrewActivity.ToString()
          + " @ " + this.calculateDateTime(timeOfUpdate).ToString());

        GeotechnicalSection              theGeotechnicalSection              =
theTunnel.GetCurrentGeotechnicalSection();
        double    Check    =    theGeotechnicalSection.GetEndChainage()    -
theTunnel.CurrentChainage;

        if ((Check <= 0.0) && (Check > -1.0)) // if this is the end of a section,
then...
        {
          ErrorLog.AddError("CheckSection first condition block entered.");
          theGeotechnicalSection.State = TunnelSectionState.Complete;
          theGeotechnicalSection.FinishTime = timeOfUpdate;
          theGeotechnicalSection.UpdateAttributeValues(timeOfUpdate);

          if          (theGeotechnicalSection.GetSectionNumber()          ==
theTunnel.GetGeotechnicalSectionCount())
          {
            theTunnel.UpdateAttributeValues(timeOfUpdate,
tunnelAttributes.State, TunnelState.Finished);
            theTunnel.UpdateAttributeValues(timeOfUpdate,
tunnelAttributes.FinishTime, theGeotechnicalSection.FinishTime);
          }
          ErrorLog.AddError("StartOneGeotechnicalSectionEvent scheduled after
one section completed. It is scheduled to occur @ "
            +     timeOfUpdate.ToString()     +     "     @     "     +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
          MyEngine.ScheduleEvent(theTunnel,
StartOneGeotechnicalSectionEvent, duration);
          theTunnel.GetTbm().UpdateAttributeValues(timeOfUpdate,
tbmAttributes.ResourceState, ResourceState.Idle);
        }
        else //ELSE: this is not the end of a section:
        {
          ErrorLog.AddError("CheckSection  second  condition  block  entered:
Schedule      ProcessWaitingTrain      Event      @      "      +
calculateDateTime(MyEngine.TimeNow + duration));
          MyEngine.ScheduleEvent(theTunnel,       ProcessWaitingTrainEvent,
duration);
        }
      }
```

```
    private void ReleaseTbm(Train theTrain)
    {
      ErrorLog.AddError("ReleaseTbm event entered @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
      ErrorLog.TraceResource("ReleaseTbm Event: RELEASE TBM BY
train");

      var resource = myTBMs[theTrain.Tunnel];
      this.MyEngine.ReleaseResource(theTrain, resource, 1);

      // the train should get track first, then start hauling.
      theTrain.WorkState = TrainWorkState.HaulingToUndercut;
      theTrain.Position = TrainLocation.Tunnel;
      theTrain.UpdateAttributeValues(MyEngine.TimeNow +
fedAmb.Lookahead);

theTrain.UnconditionalAttributeOwnershipDivestiture(theTrainAttributes1.ToArr
ay());

      ErrorLog.TraceResource("resource.InUse = " +
resource.InUse.ToString());
      ErrorLog.TraceResource("resource.ServersPreempted= " +
resource.ServersPreempted.ToString());

ErrorLog.TraceResource("++++++++++++++++++++++++++++++++++++++");
    }

    private void BreakdownEventHandler(BreakdownEntity entity)
    {
      Tunnel tunnel = entity.Tunnel;
      Resource resource = this.myTBMs[tunnel.GetObjectInstanceHandle()];
      WaitingFile file = this.myTbmFiles[tunnel.GetObjectInstanceHandle()];
      Train theTrain = getTheTrainOwningTbm(tunnel);
      this.lastChainage = nestTunnel.CurrentChainage;

      if (!this.theShift.IsWorking)
        ErrorLog.shiftReport("=BD event entered when shift is not working===
@ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
      ErrorLog.AddError("======================Breakdown=====
event entered @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
      ErrorLog.TraceResource("BreakdownEventHandler event: PREEMPT
TBM by breakdownEntity @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString() + " Logical time = " +
calculateDateTime(MyEngine.TimeNow).ToString());
      if (theTrain != null)
```

```
        {
          ErrorLog.TraceResource("                this.theTrain" + theTrain.ID +
".IsSuspended = " + theTrain.IsSuspended.ToString() + "****************");
          ErrorLog.TraceResource("                this.theTrain" + theTrain.ID +
".IsScheduled = " + theTrain.IsScheduled.ToString());
        }
        ErrorLog.TraceResource("----------------");

        // preempt resource with a priority greater than 1
        this.MyEngine.PreemptResource(this.breakdownEntity,              resource,
EmptyActionEvent, file, 2.0);

        if (theTrain != null)
        {
          ErrorLog.TraceResource("                this.theTrain" + theTrain.ID +
".IsSuspended = " + theTrain.IsSuspended.ToString() + "****************");
          ErrorLog.TraceResource("                this.theTrain" + theTrain.ID +
".IsScheduled = " + theTrain.IsScheduled.ToString());
        }
        else
        {
          ErrorLog.TraceResource("       theTrain is NULL.");
          ErrorLog.TraceResource("
resource.ServersOwnedByEntity(breakdownEntity) = "
                + resource.ServersOwnedByEntity(breakdownEntity).ToString());

        }
        ErrorLog.TraceResource("resource.ServersPreempted=            "            +
resource.ServersPreempted.ToString());
        ErrorLog.TraceResource("------------------------------");
        // TODO: see how preemption affects timing
      }

      private void RepairEventHandler(BreakdownEntity entity)
      {
        Resource                            resource                            =
this.myTBMs[entity.Tunnel.GetObjectInstanceHandle()];
        Train theTrain = getTheTrainOwningTbm(entity.Tunnel);

        if (!this.theShift.IsWorking )
          ErrorLog.shiftReport("======RepairEventHandler entered when shift
is              not              working========              "              +
this.calculateDateTime(fedAmb.CurrentTime).ToString());

        ErrorLog.AddError("=================RepairEventHandler      event
entered @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
```

```
        if (nestTunnel.CurrentChainage > this.lastChainage)
          ErrorLog.AddError("*************error message: activities going on
during TBM breakdown+++");

        ErrorLog.TraceResource("RepairEventHandler    event    entered    @
currentTime = " + this.calculateDateTime(fedAmb.CurrentTime).ToString() +
"Logical time = " + calculateDateTime(MyEngine.TimeNow).ToString());
        if (theTrain != null)
        {
          ErrorLog.TraceResource("              this.theTrain.IsSuspended = " +
theTrain.IsSuspended.ToString() + "****************");
          ErrorLog.TraceResource("              this.theTrain.IsScheduled = " +
theTrain.IsScheduled.ToString());
        }
        ErrorLog.TraceResource("----------------");

        this.MyEngine.ReleaseResource(this.breakdownEntity, resource, 1);
        if (theTrain != null)
        {
          ErrorLog.TraceResource("              this.theTrain.IsSuspended = " +
theTrain.IsSuspended.ToString() + "****************");
          ErrorLog.TraceResource("              this.theTrain.IsScheduled = " +
theTrain.IsScheduled.ToString());
        }
        ErrorLog.TraceResource("RepairEventHandler: RELEASE TBM from
preemption by BreakdownEntity ");
        ErrorLog.TraceResource("resource.ServersPreempted      =      "      +
resource.ServersPreempted.ToString());

ErrorLog.TraceResource("++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++");
      }

      private void EmptyAction(Entity entity)
      {
        ErrorLog.AddError("EmptyAction event entered");
      }

      private void ShiftOnEventHandler(Entity entity)
      {
        ErrorLog.TraceResource("Shift ON Event - release all suspended entities -
@ " + this.calculateDateTime(fedAmb.CurrentTime).ToString() );
        ErrorLog.AddError("--------------------------------------------------"       +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
```

```
         ErrorLog.AddError("enter TUnnel . Shift ON Event - release all
suspended entities ");
         ErrorLog.AddError("Tunnel.suspendedEntities.Count = " +
suspendedEntities.ToList().Count.ToString());
         //**********************
         // this part should be removed when multi-suspension working.
         if(this.suspendedEntities.ToList().Count == this.allEntities.ToList().Count)
         {
           ErrorLog.AddError("*************error message: Try to RESUME
entities consecutively twice. RESUME SKIPPED THIS TIME ************");
         }
         else
         {
         //***********END************
           ErrorLog.AddError(" +++++++++ Tunnel.ResumeEntities.Count = " +
suspendedEntities.ToList().Count.ToString());
           this.suspendedEntities.ResumeEntities(MyEngine);
           this.suspendedEntities.UnionWith(this.allEntities);
         }
       }

       private void ShiftOffEventHandler(Entity entity)
       {
         ErrorLog.TraceResource("---Shift OFF event @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString() );
         ErrorLog.shiftReport("---Enter Shift OFF event @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString() );
         ErrorLog.AddError("-------------------------------------------------- @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
         ErrorLog.AddError("---Enter TUnnel Shift OFF event");

         //**********************
         // this part should be removed when multi-suspension working.
         ErrorLog.AddError("Tunnel.suspendedEntities.Count (before suspend)= "
+ suspendedEntities.ToList().Count.ToString());
         if (this.suspendedEntities.ToList().Count < this.allEntities.ToList().Count
&& !firstShiftControl)
         {
           ErrorLog.AddError("*************error message: Try to suspend
entities consecutively twice.SUSPENSION SKIPPED THIS TIME
Tunnel************");
           //System.Windows.Forms.MessageBox.Show("Error: Attempted to
suspend entity twice.");
         }
         else
         {
```

```
        //**********END***********
            this.suspendedEntities.SuspendEntitiesIfNeed(MyEngine);
        }
        if (firstShiftControl)
        {
            firstShiftControl = false;
            this.suspendedEntities.UnionWith(this.allEntities);
            this.suspendedEntities.SuspendEntitiesIfNeed(MyEngine);
            ErrorLog.AddError("Tunnel.Enter first shift off");
        }
        ErrorLog.AddError("Tunnel.suspendedEntities.Count(after)    =    "    +
suspendedEntities.ToList().Count.ToString());
    }

    private void ProcessWaitingTrain(Tunnel theTunnel)
    {
        ErrorLog.AddError("Enter    ProcessWaitingTrain    event    @    "    +
this.calculateDateTime(fedAmb.CurrentTime).ToString());

        // check if a RequestTbm event has been delayed, then resume it.
        // only when this is not the end of one section; otherwise should wait
startSectionEvent finish.
        Train theTrain = getTheTrainAtTunnelFace(theTunnel);
        if        ((theTrain        !=        null)        &&
delayedRequestTbm[theTrain.GetObjectInstanceHandle()] == true)
        // no matter the TBM functional or not, always send the request.
Breakdown should be able to take care of it
        {
            MyEngine.ScheduleEvent(theTrain, RequestTbmEvent, 0);
            ErrorLog.AddError(" ++++ RESUME delayed RequestTbm by Train:
RequestTbmEvent_from ProcessWaitingTrain");
            delayedRequestTbm[theTrain.GetObjectInstanceHandle()] = false;
        }
        else ErrorLog.AddError(" No train is waiting for TBM @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
    }

    # endregion

        # region fedAmb Time Advance

    private        void        fedAmb_TimeAdvanceGrant(object        sender,
TimeAdvanceGrantEventArgs e)
    {
        ErrorLog.AddError("TunnelFederate TAG to " + e.theTime.ToString() + "
@ " + this.calculateDateTime(e.theTime).ToString());
```

```
            //Process any internal events that should occur at the current time.
            MyEngine.Simulate(e.theTime);

            ErrorLog.AddError("TunnelFederate NMR to " + 
MyEngine.TimeNext.ToString() + " @ " + 
this.calculateDateTime(MyEngine.TimeNext).ToString());
            //Advance time to the time of the next internal event.
            rtiAmb.NextMessageRequest(MyEngine.TimeNext);

            //ErrorLog.shiftReport("TUnnel.theShift.IsWorking = " + 
this.theShift.IsWorking.ToString() + " @ " + 
this.calculateDateTime(fedAmb.CurrentTime ).ToString());
        }

        private void fedAmb_BeginExecution(object sender, EventArgs e)
        {
            MyEngine.InitializeScenario();
            this.tunnelInfoHost.BeginExecution();

            foreach (Tunnel theTunnel in MyTunnelFactory)
            {

theTunnel.AttributeOwnershipAcquisition(theTunnelAttributes.ToArray());
               // DirtAmountPerMeter = Math.PI *theTunnel.Diameter * 
theTunnel.Diameter / 4; //Vary from diff. tunnels.

               //AdvanceRateUpdateMethods selcted = 
(AdvanceRateUpdateMethods)cbbMethod.SelectedItem;
               //theTunnel.AdvanceRateUpdateMethod = selcted;
            }

            foreach (GeotechnicalSection theGeotechnicalSection in 
MyGeotechnicalSectionFactory)
            {

theGeotechnicalSection.AttributeOwnershipAcquisition(theGeotechnicalSectionA
ttributes.ToArray());
            }

            foreach (Tbm theTbm in MyTbmFactory)
            {
               theTbm.AttributeOwnershipAcquisition(theTbmAttributes.ToArray());
            }
        }

        private void fedAmb_EndExecution(object sender, EventArgs e)
        {
           MyEngine.FinalizeScenario();
        }

        # endregion

           # region object management
        private void MyTunnelFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
        {
           var theTunnel = MyTunnelFactory[e.theObject];

           // add a new train and its waiting file for each Tunnel
           var Resource = new Resource("Train", 1);  // be changable.
           var myResourceFile = new WaitingFile("Train waiting file");

           MyEngine.Resources.Add(Resource);
           MyEngine.WaitingFiles.Add(myResourceFile);
           Resource.WaitingFiles.Add(myResourceFile);
           myTBMs.Add(e.theObject, Resource);
           myTbmFiles.Add(e.theObject, myResourceFile);

           this.allEntities.Add(theTunnel);
           this.allEntities.Add(this.breakdownEntity);
        }

        private void MyTrainFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
        {
           if (e.theValues.Contains(trainAttributes.WorkState.GetAttributeHandle()))
           {
              Train theTrain = MyTrainFactory[e.theObject];
              Tunnel t1 = theTrain.GetTunnel();
              ErrorLog.AddError("TunnelFederate Train" + theTrain.ID +
".WorkState reflect: "
                 + theTrain.WorkState.ToString()+ " @ " +
this.calculateDateTime(e.theTime).ToString());
              //THREE conditions are required to schedule capture TBM:
              //1) a train arrived tunnel face;
              //2) all activities around tunnel face have finished, including lining,
extendTrack, Survey and so on.
              //3) it has to be later than StartOneGeotechnicalSectionEvent
              // STILL, some train got missed fro mthe first two condition:
              // a) when train arrived, the tunnel has not finished all acitivities such as
survey;
              // b) when check section, the train has not arrived tunnel face yet.
```

```
            Tunnel theTunnel = MyTunnelFactory[theTrain.Tunnel];
            if (theTrain.WorkState == TrainWorkState.WaitTbm)
            {
              GeotechnicalSection                    theSection                    =
MyGeotechnicalSectionFactory[theTunnel.CurrentGeotechnicalSection];
              if
(!theTrain.IsAttributeOwnedByFederate(trainAttributes.WorkState.GetAttributeH
andle()))

theTrain.AttributeOwnershipAcquisition(theTrainAttributes1.ToArray());

              if  ((theTunnel.CrewActivity  ==  TunnelCrewActivity.WaitingTrain)
&&
                (theSection.State == TunnelSectionState.Constructing))
              {
                MyEngine.ScheduleEvent(theTrain,  RequestTbmEvent,  e.theTime
- MyEngine.TimeNow);
                ErrorLog.AddError("  +++++  RequestTbm  when  a  Train  arrived
tunnle face.@  " + this.calculateDateTime(e.theTime).ToString());
              }
              else
              {
                delayedRequestTbm[theTrain.GetObjectInstanceHandle()] = true;
                ErrorLog.AddError("++++    --------SET    delayedRequestTbm    _
RequestTbmEvent,   when   Train   ARRIVED   tunnel   face:     @       "   +
this.calculateDateTime(e.theTime).ToString());
              }
              ErrorLog.AddError("++++      theTunnel.CrewActivity      =      "      +
theTunnel.CrewActivity.ToString()
                + " @ " + this.calculateDateTime(e.theTime).ToString());
              ErrorLog.AddError("++++        theSection.State        ==        "        +
theSection.State.ToString());
            }
          }
        }

        private    void    MyTbmFactory_ReflectAttributeValues(object    sender,
ReflectAttributeValuesEventArgs e)
        {
          Tbm theTbm = MyTbmFactory[e.theObject];
          Tunnel theTunnel = this.MyTunnelFactory[theTbm.Tunnel];

          if
(e.theValues.Contains(tbmAttributes.EquipmentState.GetAttributeHandle())
&& !double.IsNaN(e.theTime))
          {
            ErrorLog.AddError("TunnelFederate Tbm.EquipmentState reflect: " +
theTbm.EquipmentState.ToString()
              + " @ " + calculateDateTime(e.theTime).ToString());
            if (theTbm.EquipmentState == EquipmentState.Down)
            {
              this.MyEngine.ScheduleEvent(new      BreakdownEntity(theTunnel),
BreakdownEvent, e.theTime - MyEngine.TimeNow);
            }
            else if (theTbm.EquipmentState == EquipmentState.Functional)
            {
              double duration = Math.Max(0, (e.theTime - fireTbmRepairDuration
- MyEngine.TimeNow));
              //make   sure   that   (e.theTime   -   fireTbmRepairDuration   -
MyEngine.TimeNow)>=0
              this.MyEngine.ScheduleEvent(new      BreakdownEntity(theTunnel),
RepairEvent, e.theTime - MyEngine.TimeNow);
            }
          }
        }

        private void MyWorkingShaftFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
        {
          // if a working shaft is finished, let the corresponding tunnel start working.
          if (e.theValues.Contains(shaftAttributes.State.GetAttributeHandle()))
          {
            Shaft theShaft = MyWorkingShaftFactory[e.theObject];
            if (theShaft.State == ShaftState.Finished)
            {
              foreach (Tunnel theTunnel in MyTunnelFactory)
              {
                if (theShaft == theTunnel.GetWorkingShaft())
                {
                  if (theTunnel.GetUndercut() == null)
                  {
                    ErrorLog.AddError("StartOneGeotechnicalSectionEvent
scheduled      in      ShaftFactory      reflect.      @      "      +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
                    MyEngine.ScheduleEvent(theTunnel,
StartOneGeotechnicalSectionEvent, e.theTime - MyEngine.TimeNow);
                  }
                  else     ErrorLog.AddError("StartOneGeotechnicalSectionEvent
until tbm installed.");
                }
              }
            }
            }
        }

        private void MyTunnelFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
        {
            Tunnel theTunnel = MyTunnelFactory[e.theObject];
            if (e.theValues.Contains(tunnelAttributes.Name.GetAttributeHandle())
                && theTunnel.Name == "NEST")
            {
                this.nestTunnel = theTunnel;
            }
        }

        private void MyTrainFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
        {
            delayedRequestTbm.Add(e.theObject, false);
            this.allEntities.Add(MyTrainFactory[e.theObject]);
        }

        private void myShiftFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
        {
            if (e.theTime > 0 &&
e.theValues.Contains(shiftAttributes.IsWorking.GetAttributeHandle()))
            {
                if (this.nestTunnel.State != TunnelState.Unstarted &&
this.nestTunnel.State != TunnelState.Finished )
                {
                    var shift = myShiftFactory[e.theObject];
                    double timeOfUpdate = Math.Max(e.theTime,
this.rtiAmb.QueryLogicalTime() + fedAmb.Lookahead);
                    if (shift.IsWorking)
                    {
                        foreach (Tbm theTbm in MyTbmFactory)
                        {
                            theTbm.UpdateAttributeValues(timeOfUpdate,
tbmAttributes.ResourceState, ResourceState.Busy);
                        }
                        ErrorLog.shiftReport("-----------------------------------------------------
--");
                        ErrorLog.shiftReport("+++ TunnelFederate.ShiftOnEvent @ " +
this.calculateDateTime(e.theTime).ToString() + " ResourceState.Busy");
                        ErrorLog.AddError("+++ TunnelFederate.Shift On Event @ " +
this.calculateDateTime(e.theTime).ToString() + " ResourceState.Busy");
```

```
                MyEngine.ScheduleEvent(this.shiftControlEntity,    ShiftOnEvent,
e.theTime - MyEngine.TimeNow);
            }
            else
            {
                ErrorLog.AddError("+++++++++++++++++    From    Tunnel:
nestTunnel.CrewActivity = " + nestTunnel.CrewActivity.ToString()
                    +          "          @          "          +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
                TimeSpan                    time                    =
this.theCalendar.StartDateTime.AddSeconds(e.theTime).TimeOfDay;
                TimeSpan target = TimeSpan.FromSeconds(this.theShift.StartTime)
+ TimeSpan.FromSeconds(this.theShift.Duration);
                Train theTrain = nestTunnel.getFirstTrain();
                if   (time   <   target   ||   (time   ==   target   &&
this.nestTunnel.CrewActivity   !=   TunnelCrewActivity.TbmOperation   &&
theTrain.CartsCounter == 0))
                // if this is not the end of a shift or all activities around the tunnel
face has been finished.
                {
                    foreach (Tbm theTbm in MyTbmFactory)
                    {
                        theTbm.UpdateAttributeValues(timeOfUpdate,
tbmAttributes.ResourceState, ResourceState.Idle);
                    }
                    ErrorLog.shiftReport("++++++          TunnelFederate.Shedule
ShiftOff   @   "   +   this.calculateDateTime(e.theTime).ToString()   +   "
ResourceState.Idle");

                    MyEngine.ScheduleEvent(this.shiftControlEntity, ShiftOffEvent,
e.theTime - MyEngine.TimeNow);

                    // Send a message of a shift off.
                    if (time == target) // if this is not the end of a shift
                    {
                        ErrorLog.AddError("+++++++++++send  an  interaction  of
finishing lining a meter = " + this.calculateDateTime(e.theTime).ToString());
                        ErrorLog.shiftReport("send an interaction of finishing lining a
meter = " + this.calculateDateTime(e.theTime).ToString());

                        double time1 = e.theTime - MyEngine.TimeNow;
                        double time2 = e.theTime - rtiAmb.QueryLogicalTime();
                        double time3 = e.theTime - fedAmb.CurrentTime;

                        this.ShiftOffTerminal.SharpOff = true;
                        this.ShiftOffTerminal.Tunnel                              =
nestTunnel.GetObjectInstanceHandle();
                        this.ShiftOffTerminal.SendInteraction(Math.Max(e.theTime,
rtiAmb.QueryLogicalTime() + fedAmb.Lookahead));
                    }
                }
                else
                {
                    ErrorLog.AddError("Tunnel Shift OFF Event not scheduled @ "
+ this.calculateDateTime(e.theTime).ToString());
                }
            }
        }
    }
}

private    void    calendarFactory_DiscoverObjectInstance(object    sender,
DiscoverObjectInstanceEventArgs e)
{
    this.theCalendar = calendarFactory[e.theObject];
}

private    void    myShiftFactory_DiscoverObjectInstance(object    sender,
DiscoverObjectInstanceEventArgs e)
{
    this.theShift = myShiftFactory[e.theObject];
}

private    void    tbmInstalledTerminal_ReceiveInteraction(object    sender,
ReceiveInteractionEventArgs e)
{
    var  theParameter  =  this.rtiAmb.GetParameterHandle(e.theInteraction,
"TailTunnel");
    if (e.theValues.Contains(theParameter))
    {
        var                        tailTunnel                              =
(TailTunnel)this.undercutFactory[(ObjectInstanceHandle)e.theValues[theParamet
er]];
        var tunnel = tailTunnel.GetTunnel();

        // Schedule the start of the first geotechnical section of the associated
tunnel.
        ErrorLog.AddError("StartOneGeotechnicalSectionEvent       scheduled
when        TBM        installed.        @        "        +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
```

```
            MyEngine.ScheduleEvent(tunnel,    StartOneGeotechnicalSectionEvent,
e.theTime - MyEngine.TimeNow);
         }
      }

      private   void   switchInstalledTerminal_ReceiveInteraction(object   sender,
ReceiveInteractionEventArgs e)
      {
         var   theParameter   =   this.rtiAmb.GetParameterHandle(e.theInteraction,
"Undercut");
         if (e.theValues.Contains(theParameter))
         {
            var                           tailTunnel                           =
(TailTunnel)this.undercutFactory[(ObjectInstanceHandle)e.theValues[theParamet
er]];

            // Schedule the start of the next geotechnical section of the associated
tunnel.
            ErrorLog.AddError("StartOneGeotechnicalSectionEvent       scheduled
when switch installed." +
               " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
            MyEngine.ScheduleEvent(tailTunnel.GetTunnel(),
StartOneGeotechnicalSectionEvent, e.theTime - MyEngine.TimeNow);
         }
      }

      private   void   myConcreteFactory_DiscoverObjectInstance(object   sender,
DiscoverObjectInstanceEventArgs e)
      {
         myConcrete = myConcreteFactory[e.theObject];
      }

      # endregion


         private double fireTbmRepairDuration
         {
             get { return this.fedAmb.Lookahead + 1.0; }
         }

         #region Private Nested Classes

      #region BreakdownEntity
      private class BreakdownEntity : Entity
      {
         public Tunnel Tunnel { get; set; }
```

```
            public BreakdownEntity(Tunnel tunnel)
            {
                this.Tunnel = tunnel;
            }
        }
        #endregion

        #endregion


    }
}
```

d. Dirt Removal Federate

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using Cosye.Framework;
using Cosye.Hla.Rti;
using Cosye.Tunneling.TunnelFederate;
using Simphony.Mathematics;
using Simphony.Simulation;

namespace Cosye.Tunneling.DirtRemoval
{
   public partial class DirtRemoval : FederateControl
   {
      #region Private Readonly Fields
         private readonly List<string> theTrainAttributes2 = new List<string>();
         private readonly List<string> theTrainAttributes1 = new List<string>();
        private readonly List<string> CraneEngeryAttributes = new List<string>();

      private readonly double hourToSeconds = 1D * 60D * 60D; // convert a hour
to seconds.

         private readonly Dictionary<Tunnel, TunnelState> LastTunnelState = new
Dictionary<Tunnel, TunnelState>();
      # endregion

      #region Private Fields


      private BasicDiscreteEventEngine MyEngine = new
BasicDiscreteEventEngine();
      private readonly Dictionary<ObjectInstanceHandle, Resource> myCranes =
new Dictionary<ObjectInstanceHandle, Resource>();
      private readonly Dictionary<ObjectInstanceHandle, WaitingFile>
waitingMyCranes = new Dictionary<ObjectInstanceHandle, WaitingFile>();
      private readonly Dictionary<ObjectInstanceHandle, Resource> myTracks =
new Dictionary<ObjectInstanceHandle, Resource>();
      private readonly Dictionary<ObjectInstanceHandle, WaitingFile>
waitingMyTracks = new Dictionary<ObjectInstanceHandle, WaitingFile>();

      private Action<Train> RequestTrackEvent;
      private Action<Train> TrackCapturedEvent;
      private Action<Train> ReturningToTunnelFaceEvent;
      private Action<Train> ArrivedTunnelFaceEvent;
private Action<Train> HaulingToUndercutEvent;
private Action<Train> RequestCraneEvent;
private Action<Train> CraneCapturedEvent;
private Action<Train> DumpingEvent;
private Action<Train> ReleaseCraneEvent;
   private Action<GeotechnicalSection> ReleaseTrackEvent;

private Weather MyWeather;
private Entity shiftControlEntity = new Entity();
private Calendar theCalendar;
private Shift theShift;
private Action<Entity> ShiftOnEvent;
private Action<Entity> ShiftOffEvent;
private Action<Entity> EmptyActionEvent;
private ShiftAwareEntities allEntities = new ShiftAwareEntities();
private ShiftAwareEntities suspendedEntities = new ShiftAwareEntities();
private Tunnel nestTunnel;
private bool firstShiftControl=true;
#endregion

   #region Public Constructors

   #region DirtRemoval()
   public DirtRemoval()
   {
       InitializeComponent();

       theTrainAttributes2.Add(trainAttributes.CurrentDirtVolume);
       theTrainAttributes2.Add(trainAttributes.CartsCounter);
       theTrainAttributes2.Add(trainAttributes.Position);
       //theTrainAttributes2.Add(trainAttributes.MuckCartCount);
       theTrainAttributes2.Add(trainAttributes.CapturedTrack);
       //theTrainAttributes2.Add(trainAttributes.EmptySpeed);
       theTrainAttributes2.Add(trainAttributes.EnergyConsumption);
       //theTrainAttributes2.Add(trainAttributes.CostPerHour);
       //theTrainAttributes2.Add(trainAttributes.LoadedSpeed);
       theTrainAttributes2.Add(trainAttributes.WorkHours);
       theTrainAttributes2.Add(trainAttributes.ResourceState);
       theTrainAttributes2.Add(trainAttributes.WorkState);

       theTrainAttributes1.Add(trainAttributes.CurrentDirtVolume);
       theTrainAttributes1.Add(trainAttributes.CartsCounter);
       theTrainAttributes1.Add(trainAttributes.EnergyConsumption);
       theTrainAttributes1.Add(trainAttributes.Position);
       theTrainAttributes1.Add(trainAttributes.WorkState);
```

```
            CraneEngeryAttributes.Add(craneAttributes.EmissionFactor);
            CraneEngeryAttributes.Add(craneAttributes.EnergyConsumption);
            CraneEngeryAttributes.Add(craneAttributes.WorkHours);

            // Initialize events.
            RequestTrackEvent = new Action<Train>(RequestTrack);
            TrackCapturedEvent = new Action<Train>(TrackCaptured);
            ReturningToTunnelFaceEvent                    =                    new
Action<Train>(ReturningToTunnelFace);
        ArrivedTunnelFaceEvent = new Action<Train>(ArrivedTunnelFace);
            HaulingToUndercutEvent = new Action<Train>(HaulingToUndercut);
            RequestCraneEvent = new Action<Train>(CaptureCrane);
            CraneCapturedEvent = new Action<Train>(CraneCaptured);
            DumpingEvent = new Action<Train>(DumpingAndLoading);
            ReleaseCraneEvent = new Action<Train>(ReleaseCrane);
            ReleaseTrackEvent                    =                    new
Action<GeotechnicalSection>(ReleaseTrack);

        ShiftOffEvent = new Action<Entity>(ShiftOffEventHandler);
        ShiftOnEvent = new Action<Entity>(ShiftOnEventHandler);
        EmptyActionEvent = new Action<Entity>(EmptyAction);

            MyEngine.InitializeEngine();
        }
        #endregion

        #endregion

      #region Private Methods
      #region calculateDateTime(double)
      private DateTime calculateDateTime(double logicTime)
      {
        if (logicTime > 0 && !double.IsInfinity(logicTime))
           return this.theCalendar.StartDateTime.AddSeconds(logicTime);
        else return DateTime.MinValue;
      }
      #endregion

        #region EnergyConsumptionRateOfTrain(Train)
        private double EnergyConsumptionRateOfTrain(Train theTrain)
        {
            double Rate = 0;

            //Train : 1.704 * weight of soil (t) * the traveling distance(ft)/100
            // 1.704 * soil weight(t)*D(m)* 3.28 (ft/m) /100
            if (theTrain.WorkState == TrainWorkState.HaulingToUndercut) //
hauling means moving from tunnel face to undercut.
            {
                // average dirt density = 120 lb/cubic feet =   1.922 (t/cubic meter)
                // the weight of one locomotive = 5897 kilograms = 5.897 t; the
weight of one empty muck car = 1 t
                // 1 m = 3.28 feet
                Rate = 1.704 * (theTrain.CurrentDirtVolume * 1.922 + 5.897 +
theTrain.MuckCartCount) * 3.28 / 100 / 1000; // KWh /m
            }
            if (theTrain.WorkState == TrainWorkState.ReturningToTbm)
            {
                Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
                double    linerWeight    =    (theTunnel.CurrentChainage    <
theTunnel.GetTbm().GantryLength ? 1 / theTrain.MuckCartCount : 1) * 4.8; // the
weight of one liner (four segments) : 4.8 t
                Rate = 1.704 * (linerWeight + 5.897 + theTrain.MuckCartCount) *
3.28 / 100 / 1000; // KWh /m
            }
            ////  200 KWh per hour for working (hauling and returning),  20 KWh
per hour for Idling
            //if (theTrain.WorkState == Tr);ainWorkState.Hauling)
            //{
            //    Rate = 200 / 3600D;
            //}
            //else if (theTrain.WorkState == TrainWorkState.ReturningToTbm)
            //{
            //    Rate = 200D / 3600D;
            //}
            //else
            //{
            //    Rate = 20D / 3600D;
            //}
            return Rate;
        }
        #endregion

        #region InitializeInterface(Train)
        private void InitializeInterface(Train theTrain)
        {
            // Initialize the interface.
            Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
            txtTunnelName.Text = theTunnel.Name.ToString();
            txtTrainID.Text = theTrain.ID.ToString();

            txtTrainStatus.Text = theTrain.WorkState.ToString();
            // "Unstarted";

            this.Refresh();
        }
        #endregion

        #region UpdateMyInterface(Train)
        private void UpdateMyInterface(Train theTrain)
        {
            // Update Interface.
            txtTrainID.Text = theTrain.ID.ToString();
            txtTrainLocation.Text = theTrain.Position.ToString();
            txtDirtAmount.Text = theTrain.CurrentDirtVolume.ToString("0.00");
            txtLogicalTime.Text            =            (MyEngine.TimeNow            /
hourToSeconds).ToString("0.00");
            txtTrainStatus.Text = theTrain.WorkState.ToString();

            //double    pro    =    Math.Round(100    *    theTrain.CurrentLoad    /
theTrain.CapacityPerCart / theTrain.NumberOfMuckCars);
            //LoadingProgressBar.Value = System.Convert.ToInt16(pro);
            //txtWindSpeed.Text = MyWeather.SpeedOfMaxGust.ToString();

            this.Refresh();
            //this.txtDirtAmount.TextChanged              +=              new
System.EventHandler(this.txtDirtAmount_TextChanged);
        }
        #endregion

        #region getFirstTrain(Tunnel)
        private Train getFirstTrain(Tunnel theTunnel)
        {
            Train currentTrain = null;
            foreach (Train theTrain in MyTrainFactory)
            {
                if (theTrain.Tunnel == theTunnel.GetObjectInstanceHandle())
                {
                        if (currentTrain == null) currentTrain = theTrain;
                        else if (currentTrain.ID > theTrain.ID)
                        {
                                currentTrain = theTrain;
                        }
                }
            }
            return currentTrain;
        }
        #endregion
```

```
#region getSecondTrain(Tunnel)
private Train getSecondTrain(Tunnel theTunnel)
{
    Train currentTrain = null;
    foreach (Train theTrain in MyTrainFactory)
    {
        if (theTrain.Tunnel == theTunnel.GetObjectInstanceHandle())
        {
                if (currentTrain == null) currentTrain = theTrain;
                else if (currentTrain.ID < theTrain.ID)
                {
                        currentTrain = theTrain;
                }
        }
    }
    return currentTrain;
}
#endregion

#region LoadingDuration(Train)
private double LoadingDuration(Train theTrain)
{
    double duration = theTrain.MaterialLoadingDuration.Sample();

    if (MyWeather.SpeedOfMaxGust > 35)
    {
        duration += duration * (MyWeather.SpeedOfMaxGust / 120);
    }
    if (MyWeather.MinTemp < -25)
    {
        duration = duration * (-MyWeather.MinTemp / 25);
    }
    return duration;
}
#endregion

#region UnloadingDuration(Train)
private double UnloadingDuration(Train theTrain)
{
    double duration = theTrain.MuckCartDumpDuration.Sample();

    if (MyWeather.SpeedOfMaxGust > 35)
    {
        duration += duration * (MyWeather.SpeedOfMaxGust / 120);
    }
```

```
                if (MyWeather.MinTemp < -25)
                {
                    duration = duration * (-MyWeather.MinTemp / 25);
                }
                return duration;
            }
            #endregion

            #region ERofCrane(Crane)
            private double ERofCrane(Crane theCrane)
            {
                double Rate = 16.82 / 3600;  // operation time is in second.
                //Energy consumption = operation time (hr) * 16.82 (Kg/hr)

                return Rate;
            }
            #endregion

        #endregion

        #region Private Events

            #region CaptureTrack(Train)
            private void RequestTrack(Train theTrain)
            {
                ErrorLog.AddError("DirtRemoval.CaptureTrack event entered: Train"
+ theTrain.ID
               + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
                var theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
                var resource = myTracks[theTunnel.GetObjectInstanceHandle()];
                var file = waitingMyTracks[theTunnel.GetObjectInstanceHandle()];
                this.MyEngine.RequestResource(theTrain,           resource,           1,
TrackCapturedEvent, file);
                UpdateMyInterface(theTrain);
            }
            #endregion

            #region TrackCaptured(Train)
            private void TrackCaptured(Train theTrain)
            {
            ErrorLog.AddError("DirtRemoval.TrackCaptured event entered: Train" +
theTrain.ID
               + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
                MyEngine.ScheduleEvent(theTrain, ReturningToTunnelFaceEvent, 0);
                theTrain.WorkState = TrainWorkState.ReturningToTbm;
                theTrain.Position = TrainLocation.Tunnel;
                theTrain.CapturedTrack = true;        // showing the train  captured a
track
                theTrain.UpdateAttributeValues(MyEngine.TimeNow                +
fedAmb.Lookahead);
                UpdateMyInterface(theTrain);
            }
            #endregion

        #region ReturningToTunnelFace(Train)
        private void ReturningToTunnelFace(Train theTrain)
            {
            ErrorLog.AddError("DirtRemoval.ReturningToTunnelFace event entered:
Train"      +      theTrain.ID      +      "      @      "      +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
                Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
            double duration = Math.Max(Math.Log10(theTunnel.CurrentChainage) *
60D, fedAmb.Lookahead); //theTunnel.CurrentChainage / theTrain.EmptySpeed ;
            MyEngine.ScheduleEvent(theTrain, ArrivedTunnelFaceEvent, duration -
fedAmb.Lookahead);
            }
            #endregion

        #region ArrivedTunnelFace(Train)
        private void ArrivedTunnelFace(Train theTrain)
            {
            ErrorLog.AddError("DirtRemoval.ArrivedTunnelFace    event    entered:
Train"      +      theTrain.ID      +      "      @      "      +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
                Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
            double duration = fedAmb.Lookahead;
                theTrain.EnergyConsumption    +=    theTunnel.CurrentChainage    *
EnergyConsumptionRateOfTrain(theTrain);  //  CurrentChainage  means  the
traveling distance of the train? the usnit of CurrentChainage is "m"?

                theTrain.Position = TrainLocation.TunnelFace;
                theTrain.WorkState = TrainWorkState.WaitTbm;

                theTrain.UpdateAttributeValues(MyEngine.TimeNow + duration);
                UpdateMyInterface(theTrain);

       theTrain.UnconditionalAttributeOwnershipDivestiture(theTrainAttributes1.To
Array());
            }
            #endregion

        #region HaulingToUndercut(Train)
```

```
      private void HaulingToUndercut(Train theTrain)
         {
         ErrorLog.AddError("DirtRemoval.HaulingToUndercut    event    entered:
Train " + theTrain.ID
            + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
             Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
            double duration = Math.Max(Math.Log10(theTunnel.CurrentChainage)
*      60D,      fedAmb.Lookahead);             //theTunnel.CurrentChainage
/theTrain.LoadedSpeed ;
             theTrain.EnergyConsumption     +=    theTunnel.CurrentChainage     *
EnergyConsumptionRateOfTrain(theTrain);

             if (duration < fedAmb.Lookahead) duration = fedAmb.Lookahead;

     theTrain.AttributeOwnershipAcquisition(theTrainAttributes1.ToArray());
             theTrain.Position = TrainLocation.Undercut;
             theTrain.WorkState = TrainWorkState.WaitCrane;

             GeotechnicalSection                    theSection                    =
MyGeotechnicalSectionFactory[theTunnel.CurrentGeotechnicalSection];
             this.MyEngine.ScheduleEvent(theSection,          ReleaseTrackEvent,
duration);

             theTrain.UpdateAttributeValues(MyEngine.TimeNow + duration);
             UpdateMyInterface(theTrain);

             MyEngine.ScheduleEvent(theTrain, RequestCraneEvent, duration);
         }
         #endregion

         #region CaptureCrane(Train)
         private void CaptureCrane(Train theTrain)
         {
             ErrorLog.AddError("DirtRemoval.CaptureCrane event entered: Train "
+ theTrain.ID
            + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
             var theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
             var theWorkingShaft = theTrain.GetWorkingShaft();
             //var myResource = myTracks[theTunnel.GetObjectInstanceHandle()];
             //theTrain.ReleaseResource(myResource, 1);

             var                          resource                          =
myCranes[theWorkingShaft.GetObjectInstanceHandle()];
             var                          myFile                            =
waitingMyCranes[theWorkingShaft.GetObjectInstanceHandle()];
```

```
            this.MyEngine.RequestResource(theTrain,            resource,            1,
CraneCapturedEvent, myFile);
        }
        #endregion

        #region CraneCaptured(Train)
        private void CraneCaptured(Train theTrain)
        {
            ErrorLog.AddError("DirtRemoval.CraneCaptured event entered: Train
" + theTrain.ID
           + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
            Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
            //theTrain.CurrentLoad = 0;
            theTrain.WorkState = TrainWorkState.Dumping;

            theTrain.UpdateAttributeValues(MyEngine.TimeNow                    +
fedAmb.Lookahead);
            UpdateMyInterface(theTrain);
            MyEngine.ScheduleEvent(theTrain, DumpingEvent, 0);
        }
        #endregion

        #region DumpingAndLoading(Train)
        private void DumpingAndLoading(Train theTrain)
        {
           ErrorLog.AddError("DirtRemoval.DumpingAndLoading event entered:
Train " + theTrain.ID
           + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
            Tunnel theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
            double duration = LoadingDuration(theTrain);
            if (theTunnel.State == TunnelState.Startup)
            {
                duration += UnloadingDuration(theTrain);
            }
            else
            {
                for (int count = 1; count < theTrain.MuckCartCount; count++)
                {
                        duration += UnloadingDuration(theTrain);
                }
            }
            theTrain.EnergyConsumption                                    +=
EnergyConsumptionRateOfTrain(theTrain) * duration;

            Crane theCrane = theTrain.GetCranes()[0];
            theCrane.EnergyConsumption += ERofCrane(theCrane) * duration;
            theCrane.UpdateAttributeValues(MyEngine.TimeNow + duration);

            theTrain.CurrentDirtVolume = 0;
            theTrain.WorkState = TrainWorkState.WaitTunnelTrack;
            theTrain.UpdateAttributeValues(MyEngine.TimeNow + duration);
            UpdateMyInterface(theTrain);
            MyEngine.ScheduleEvent(theTrain, ReleaseCraneEvent, duration);
        }
        #endregion

        #region ReleaseCrane(Train)
        private void ReleaseCrane(Train theTrain)
        {
        ErrorLog.AddError("DirtRemoval.ReleaseCrane event entered: Train_ " +
theTrain.ID
           + " @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
            var theWorkingShaft = theTrain.GetWorkingShaft();

            var crane = myCranes[theWorkingShaft.GetObjectInstanceHandle()];
            if (crane.Available == 0) this.MyEngine.ReleaseResource(theTrain,
crane, 1);

        //GeotechnicalSection                    theSection                    =
theTunnel.GetCurrentGeotechnicalSection();
        //if (theSection.State != TunnelSectionState.Complete)
        var theTunnel = this.MyTunnelFactory[theTrain.Tunnel];
        if ( theTunnel.State != TunnelState.Finished)
            {
           ErrorLog.AddError("CaptureTrackEvent            scheduled            in
DirtRemoval.ReleaseCrane");
           MyEngine.ScheduleEvent(theTrain, RequestTrackEvent, 0);
            }
        }
        #endregion

        #region ReleaseTrack(GeotechnicalSection)
        private void ReleaseTrack(GeotechnicalSection theSection)
        {
        ErrorLog.AddError("DirtRemoval.ReleaseTrack event entered" + " @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());
            var theTunnel = MyTunnelFactory[theSection.Tunnel];
            var resource = myTracks[theTunnel.GetObjectInstanceHandle()];

            foreach (Train theTrain in this.MyTrainFactory)
            {
                if (theTrain.Tunnel == theTunnel.GetObjectInstanceHandle())
                {
                        if (resource.ServersOwnedByEntity(theTrain) >= 1)
                        {
                                ErrorLog.AddError("DirtRemoval.ReleaseTrack
event conditions passed for Train" + theTrain.ID);
                                this.MyEngine.ReleaseResource(theTrain, resource,
1);
                theTrain.CapturedTrack = false; // showing the train will not keep a
track
                theTrain.UpdateAttributeValues(fedAmb.CurrentTime              +
fedAmb.Lookahead);
                                break;
                        }
                }
            }
        }
        #endregion

     # region ShiftOnEventHandler
     /// <summary>
     /// try to resume all suspended entities when shift changes to ON
     /// </summary>
     /// <param name="entity"></param>
     private void ShiftOnEventHandler(Entity entity)
     {
        ErrorLog.AddError("----------------------------------------------------");
        ErrorLog.AddError("++ enter DirtRemoval.Shift ON Event - release all
suspended entities - @ " + calculateDateTime(MyEngine.TimeNow).ToString());
        //this.suspendedEntities.ResumeEntities(MyEngine);
        //this.suspendedEntities.UnionWith(this.allEntities);

        //***********************
        // this part should be removed when multi-suspension working.
        if              (this.suspendedEntities.ToList().Count              ==
this.allEntities.ToList().Count)
           ErrorLog.AddError("*************error message: Try to RESUME
entities consecutively twice. SKIPPED this time. DirtRemoval************");
        else
        {
           //***********END************
           ErrorLog.AddError(" +++++++++ DirtRemoval.ResumeEntities.Count
= " + suspendedEntities.ToList().Count.ToString());
           this.suspendedEntities.ResumeEntities(MyEngine);
           this.suspendedEntities.UnionWith(this.allEntities);
        }
     }
```

```
    #endregion

    # region ShiftOffEventHandler
    /// <summary>
    /// try to suspend all scheduled entities when shift changes to OFF
    /// </summary>
    /// <param name="entity"></param>
    private void ShiftOffEventHandler(Entity entity)
    {
      ErrorLog.AddError("--------------------------------------------------");
      ErrorLog.AddError("---++ Enter DirtRemoval.Shift OFF event @ " +
calculateDateTime(MyEngine.TimeNow).ToString());
      //this.suspendedEntities.SuspendEntitiesIfNeed(MyEngine);
      //ErrorLog.AddError("DirtRemoval.suspendedEntities.Count = " +
suspendedEntities.ToList().Count.ToString());

      //***********************
      // this part should be removed when multi-suspension working.
      ErrorLog.AddError("DirtRemoval.suspendedEntities.Count (before
suspend)= " + suspendedEntities.ToList().Count.ToString());
      if (this.suspendedEntities.ToList().Count < this.allEntities.ToList().Count
&& !firstShiftControl)
      {
        ErrorLog.AddError("*************error message: Try to SUSPEND
entities consecutively twice. SKIPPED this time. DirtRemoval************");
        //System.Windows.Forms.MessageBox.Show("Error: Attempted to
suspend entity twice.");
      }
      else
      {
        //***********END************
        this.suspendedEntities.SuspendEntitiesIfNeed(MyEngine);
      }
      if (firstShiftControl)
      {
        firstShiftControl = false;
        this.suspendedEntities.UnionWith(this.allEntities);
        this.suspendedEntities.SuspendEntitiesIfNeed(MyEngine);
        ErrorLog.AddError("DirtRemoval.Enter first shift off");
      }
      ErrorLog.AddError("DirtRemoval.suspendedEntities.Count(after) = " +
suspendedEntities.ToList().Count.ToString());
    }
    #endregion

    private void EmptyAction(Entity entity)
```

```
        {
          ErrorLog.AddError("DirtRemoval _ EmptyAction event entered _ track
captured by shifteEntity @ "
              + this.calculateDateTime(MyEngine.TimeNow));

        }
        #endregion

        # region Private Object Management

        #region                 MyShaftGeneralFactory_DiscoverObjectInstance(object,
DiscoverObjectInstanceEventArgs)
        private void MyShaftGeneralFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
          {
              // add a new crane and its waiting file for each working shaft
              var theShaft = MyShaftFactory[e.theObject];
              //if (theShaft.Type == ShaftType.WorkingShaft)
              //{
              var myCrane = new Resource("Crane", 1);  // be changable.
              var myCraneFile = new WaitingFile("Crane File");
              MyEngine.Resources.Add(myCrane);
              MyEngine.WaitingFiles.Add(myCraneFile);
              myCrane.WaitingFiles.Add(myCraneFile);
              myCranes.Add(e.theObject, myCrane);
              waitingMyCranes.Add(e.theObject, myCraneFile);
              //}

          }
          #endregion

          #region                 MyTunnelFactory_DiscoverObjectInstance(object,
DiscoverObjectInstanceEventArgs)
          private void MyTunnelFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
          {
              // add a track as a resource and its waiting file for each Tunnel
              var myTrack = new Resource("Track", 1);  // be changable.
              var myTrackFile = new WaitingFile("Track File");
              MyEngine.Resources.Add(myTrack);
              MyEngine.WaitingFiles.Add(myTrackFile);
              myTrack.WaitingFiles.Add(myTrackFile);
              myTracks.Add(e.theObject, myTrack);
              waitingMyTracks.Add(e.theObject, myTrackFile);
          }
          #endregion
          #region                 MyTunnelFactory_ReflectAttributeValues(object, 
ReflectAttributeValuesEventArgs)
          private void MyTunnelFactory_ReflectAttributeValues(object sender, 
ReflectAttributeValuesEventArgs e)
          {
          Tunnel theTunnel = MyTunnelFactory[e.theObject];
          if (e.theValues.Contains(tunnelAttributes.Name.GetAttributeHandle())
              && theTunnel.Name == "NEST")
          {
              this.nestTunnel = theTunnel;
          }

          if (e.theValues.Contains(tunnelAttributes.State.GetAttributeHandle()))
             {
              ErrorLog.AddError("DirtRmoval    Tunnel.State    reflect:    "    + 
theTunnel.State.ToString()
             + "  @ " + this.calculateDateTime(fedAmb.CurrentTime).ToString());
               if (!LastTunnelState.ContainsKey(theTunnel))
                       LastTunnelState.Add(theTunnel, TunnelState.Unstarted);
               if (LastTunnelState[theTunnel] != theTunnel.State)
               {
                       ErrorLog.AddError("DirtRmoval    Tunnel.State    reflect 
conditions passed");
                       LastTunnelState[theTunnel] = theTunnel.State;
                       Train theTrain = null;
                       if                  (theTunnel.State                  == 
TunnelState.MultipleTrainConstructing)
                       {
                               theTrain = getSecondTrain(theTunnel);
                       }
                       else if (theTunnel.State == TunnelState.Startup)
                       {
                               theTrain = getFirstTrain(theTunnel);
                               InitializeInterface(theTrain);
                       }

             // initialize the first train event at correct time for both trains.
                       if (theTrain != null)
                       {
                               // let a train start working at an appropriate time.
                               theTrain.CurrentDirtVolume = 0;
                               theTrain.WorkState                              = 
TrainWorkState.WaitTunnelTrack;
                               // do not consider train breakdown, so do not use 
euipmentState.
```

```
                              if (theTunnel.State != TunnelState.Finished)
                              {
                         ErrorLog.AddError("DirtRemoval.RequestTrackEvent
scheduled in DirtRemoval Tunnel.State reflect");
                                   MyEngine.ScheduleEvent(theTrain,
RequestTrackEvent, e.theTime - MyEngine.TimeNow);
                              }
                              theTrain.UpdateAttributeValues(e.theTime           +
fedAmb.Lookahead);
                         }
                    }
               }
               if
(e.theValues.Contains(tunnelAttributes.CrewActivity.GetAttributeHandle()))
               {
                    ErrorLog.AddError("DirtRemoval Tunnel.CrewActivity reflect: "
                 + theTunnel.CrewActivity + " @ " + calculateDateTime(e.theTime));
               }
          }
          #endregion

          #region                    tbmFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
          private     void     tbmFactory_ReflectAttributeValues(object     sender,
ReflectAttributeValuesEventArgs e)
          {
               if
(e.theValues.Contains(tbmAttributes.WorkState.GetAttributeHandle()))
               {
                    Tbm theTbm = this.tbmFactory[e.theObject];
                    ErrorLog.AddError("DirtRemoval   Tbm.WorkState   reflect:   "   +
theTbm.WorkState.ToString()
                 + " @ "+ calculateDateTime(e.theTime).ToString());
               }
          }
          #endregion

          #region                 MyTrainFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
          private   void   MyTrainFactory_ReflectAttributeValues(object   sender,
ReflectAttributeValuesEventArgs e)
          {
               // hauling train, if it is loaded.
               if
(e.theValues.Contains(trainAttributes.WorkState.GetAttributeHandle()))
               {
                Train theTrain = MyTrainFactory[e.theObject];
            ErrorLog.AddError("DirtRemoval Train" + theTrain.ID + " WorkState
reflect: "
            +      theTrain.WorkState.ToString()      +      "      @      "      +
calculateDateTime(e.theTime).ToString());
                if (theTrain.WorkState == TrainWorkState.HaulingToUndercut)
                {
                    MyEngine.ScheduleEvent(theTrain,
HaulingToUndercutEvent, e.theTime - MyEngine.TimeNow);
                }
            }
        }
        #endregion

        #region            MyWeatherFactory_DiscoverObjectInstance(object,
DiscoverObjectInstanceEventArgs)
        private void MyWeatherFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
        {
            MyWeather = MyWeatherFactory[e.theObject];
        }
        #endregion

    #endregion

    # region Time Management

        #region                              fedAmb_TimeAdvanceGrant(object,
TimeAdvanceGrantEventArgs)
        private      void      fedAmb_TimeAdvanceGrant(object      sender,
TimeAdvanceGrantEventArgs e)
        {
        ErrorLog.AddError("DirtRemoval TAG to " + e.theTime.ToString() + " @
" + this.calculateDateTime(e.theTime).ToString());
            //Process any internal events that should occur at the current time.
            MyEngine.Simulate(e.theTime);

        ErrorLog.AddError("DirtRemoval      NMR      to      "      +
MyEngine.TimeNext.ToString()      +      "      @      "      +
this.calculateDateTime(MyEngine.TimeNext).ToString());
            //Advance time to the time of the next internal event.
            rtiAmb.NextMessageRequest(MyEngine.TimeNext);
        }
        #endregion

        #region fedAmb_BeginExecution(object, EventArgs)
        private void fedAmb_BeginExecution(object sender, EventArgs e)
        {
            MyEngine.InitializeScenario();

            foreach (Train theTrain in MyTrainFactory)
            {

    theTrain.AttributeOwnershipAcquisition(theTrainAttributes2.ToArray());
                //MyEngine.ScheduleEvent(theTrain, CaptureTrackEvent, 0);
            }

            foreach (Crane theCrane in MyCraneFactory)
            {

    theCrane.AttributeOwnershipAcquisition(CraneEngeryAttributes.ToArray());
            }
        }
        #endregion

        #region fedAmb_EndExecution(object, EventArgs)
        private void fedAmb_EndExecution(object sender, EventArgs e)
        {
            MyEngine.FinalizeScenario();
        }
        #endregion

      private    void    calendarFactory_DiscoverObjectInstance(object    sender,
DiscoverObjectInstanceEventArgs e)
      {
         this.theCalendar = calendarFactory[e.theObject];
      }

      #endregion

      private  void  MyGeotechnicalSectionFactory_DiscoverObjectInstance(object
sender, DiscoverObjectInstanceEventArgs e)
      {
         this.allEntities.Add(MyGeotechnicalSectionFactory[e.theObject]);
      }

      private    void    MyTrainFactory_DiscoverObjectInstance(object    sender,
DiscoverObjectInstanceEventArgs e)
      {
         this.allEntities.Add(MyTrainFactory[e.theObject]);
      }
      private void shiftFactory_DiscoverObjectInstance(object sender,
DiscoverObjectInstanceEventArgs e)
      {
        this.theShift = myShiftFactory[e.theObject];
      }

      private void shiftFactory_ReflectAttributeValues(object sender,
ReflectAttributeValuesEventArgs e)
      {
        var shift = myShiftFactory[e.theObject];
        if (e.theValues.Contains(shiftAttributes.IsWorking.GetAttributeHandle()))
        {
          if (this.nestTunnel != null && this.nestTunnel.State !=
TunnelState.Unstarted && this.nestTunnel.State != TunnelState.Finished)
          {
            double timeOfUpdate = Math.Max(e.theTime,
this.rtiAmb.QueryLogicalTime() + fedAmb.Lookahead);
            if (shift.IsWorking)
            {
                MyEngine.ScheduleEvent(this.shiftControlEntity, ShiftOnEvent,
e.theTime - MyEngine.TimeNow);
            }
            else
            {
                ErrorLog.AddError("++++++++++++++++++ From DirtRemoval:
nestTunnel.CrewActivity = " + nestTunnel.CrewActivity.ToString()
              + " @ " +
this.calculateDateTime(fedAmb.CurrentTime).ToString());

                TimeSpan time =
this.theCalendar.StartDateTime.AddSeconds(this.rtiAmb.QueryLogicalTime()).Ti
meOfDay;
                TimeSpan target = TimeSpan.FromSeconds(this.theShift.StartTime)
+ TimeSpan.FromSeconds(this.theShift.Duration);
                Train theTrain = nestTunnel.getFirstTrain();
                if (time < target)
                {
              MyEngine.ScheduleEvent(this.shiftControlEntity, ShiftOffEvent,
e.theTime - MyEngine.TimeNow);
                }
            }
          }
        }
      }
      private void ShiftOffTerminal_ReceiveInteraction(object sender,
ReceiveInteractionEventArgs e)
      {
        var Parameter1 = this.rtiAmb.GetParameterHandle(e.theInteraction,
"Tunnel");
        var parameter2 = this.rtiAmb.GetParameterHandle(e.theInteraction,
"SharpOff");

        if (e.theValues.Contains(parameter2))
        {
          MyEngine.ScheduleEvent(this.shiftControlEntity, ShiftOffEvent,
e.theTime - MyEngine.TimeNow);
        }
      }
    }
}
```

e. Scenario Setup Federate

```
namespace Cosye.Tunneling.ScenarioSetup
{
    using System;
    using System.Collections.Generic;
    using System.Data;
    using System.Data.SqlClient;
    using System.Diagnostics;
    using System.IO;
    using System.Linq;
    using System.Runtime.Serialization.Formatters.Binary;
    using System.Windows.Forms;
    using System.Windows.Forms.DataVisualization.Charting;
    using Cosye.Framework;
    using Cosye.Hla.Rti;
    using Cosye.Tunneling;
    using Simphony;
    using Simphony.ComponentModel;
    using Simphony.Mathematics;
    using Simphony.Simulation;
    using Simphony.Windows.Forms;
    using Calendar = Cosye.Tunneling.Calendar;

    public partial class ScenarioSetup : FederateControl
    {
        #region Private Readonly Fields

        private readonly List<string> theShaftAttributes = new List<string>();
        private readonly List<string> theShaftSectionAttributes = new
List<string>();
        private readonly List<string> theUndercutAttributes = new List<string>();
        private readonly List<string> theTailTunnelSectionAttributes = new
List<string>();

        private readonly List<string> theTunnelAttributes = new List<string>();
        private readonly List<string> theGeotechnicalSectionAttributes = new
List<string>();
        private readonly List<string> theGeometrySectionAttributes = new
List<string>();

        private readonly List<string> theTrainAttributes = new List<string>();
        private readonly List<string> theCraneAttributes = new List<string>();

        private readonly List<string> theTbmAttributes = new List<string>();
        private readonly List<string> theScenarioAttributes = new List<string>();
        private readonly List<string> theProjectAttributes = new List<string>();

        private double[] GAMEgeotechnicalSections;
        private double[] GAMEgeometrySections;

        private readonly double[] NESTGeotechnicalSections = new double[]
{ 50.0, 159D, 150D, 250D, 1050D, 1200D, 350D, 538D };
        private readonly double[] NESTGeometrySections = new double[]
{ 310.36, 115.1, 525.86, 961D, 640D, 1144.97 };

        private readonly double[] ShaftSectionLength = new double[] { 13.54 };

        private List<RectangularShaftSection> shaftSections;

        // General Setting of a Scenario.
        const int TotalRectWorkingShaft = 1;
        const int TotalTunnel = 1;
        const int TotalShaftSection = 1;
        const int TotalTbm = 1;  // shound not be more than max (totaltunnel,
totalshaft)
        const int TotalTrain = 2;
        const int TotalMuckCars = 6;
        const double ShaftSectionDepth = 1;

        // Keep track of the waiting times of the TBM and the trains.
        private readonly Dictionary<ObjectInstanceHandle, TimeKeeper>
trainDownTimes;
        private readonly Dictionary<ObjectInstanceHandle, TimeKeeper>
tbmDownTimes;

        // Chart Collectors
       private readonly Simphony.Modeling.Chart<DateTime> tbmRateCollector;
        private readonly Simphony.Modeling.Chart<DateTime>
tbmCumRateCollector;
        private readonly Simphony.Modeling.Chart<DateTime>
workingShaftProgressCollector;
        private readonly Simphony.Modeling.Chart<DateTime>
workingShaftCostCollector;
        private readonly Simphony.Modeling.Chart<DateTime>
tunnelProgressCollector;
        private readonly Simphony.Modeling.Chart<DateTime>
tunnelCostCollector;

        #endregion
        #region Private Fields
        private RectangularShaft theShaft;
        private ConcreteLinerInventory myConcrete;
        private Tunnel theTunnel;
        private int trainBreakdownCount;
        private List<double> tbmBreakdownsTimes;
        private List<double> tbmRepairTimes;
        private DateTime periodStartDate;
        private DateTime periodEndDate;
        private DateTime scenarioStartDate;
        private double previousChainage;
        private double previousShaftCost;
        private double previousShaftIncome;
        private double previousShaftProgress;
        private double previousTunnelCost;
        private double previousTunnelIncome;
        private double previousTunnelProgress;
        private bool isStarted;
        private bool isGame;
        #endregion

        #region Private Readonly Distributions

        #region Breakdown and Repair
        private readonly Distribution DisBreakdown = new Triangular(2000D *
60D * 60D, 3000D * 60D * 60D, 2200D * 60D * 60D);
        private readonly Distribution DisRepair = new Triangular(5D * 60D *
60D, 20D * 60D * 60D, 15D * 60D * 60D);
        #endregion

        #region Shaft
        private readonly Distribution onePileDrivingTime = new Triangular(3D *
60D * 60D, 4.5 * 60D * 60D, 4 * 60D * 60D);
        private readonly Distribution shaftMachineExcavationRate = new
Triangular(50D / 3600D, 80D / 3600D, 70D / 3600D);
        private readonly Distribution shaftHandExcavationRate = new
Triangular(2.0 / 3600.0, 4.0 / 3600.0, 3.0 / 3600.0);
        #endregion

        #region Tunnel
        // time to line one ring -> Tbm.LiningTime
        private readonly Distribution DisLining = new Beta(15, 25, 2 * 60.0, 5 *
60.0);
        // time to reset one liner length -> Tbm.ResettingTime
        private readonly Distribution DisResetting = new Uniform(2 * 60D, 4 *
60D);
        // time to extend the utility(ies) -> Tunnel.ExtendUtilityTime
        private readonly Distribution DisExtendUtility = new Triangular(10D *
60D, 20D * 60D, 15D * 60D);
        // time to extend the train track -> Tunnel.ExtendTrackTime
        private readonly Distribution DisExtendTrack = new Triangular(0.5D *
60D * 60D, 2D * 60D * 60D, 1D * 60D * 60D);
        // time to survey the tunnel -> GeometrySection.SurveyTime
        private readonly Distribution DisSurvey = new Triangular(1.5D * 60D *
60D, 2D * 60D * 60D, 1.8D * 60D * 60D);
        // time to extend gantry -> Tbm.GantryTime (might also want to make
ConveyorTime into a Distribution)
        private readonly Distribution DisExtendGantry = new Triangular(2.5D *
60D * 60D, 4D * 60D * 60D, 3.5D * 60D * 60D);
        // time to install the TBM -> Tbm.InstallationTime
        private readonly Distribution tbmInstallTime = new Triangular(22D *
3600D, 30D * 3600D, 26D * 3600D);
        // time to install the switch -> TailTunnel.SwitchInstallationTime
        private readonly Distribution switchInstallTime = new Triangular(52D *
3600D, 60D * 3600D, 56D * 3600D);
        // time to load one train material cart
        private readonly Distribution DisLoading = new Beta(2, 2, 3 * 60, 7 * 60);
        // time to dump one train muck cart
       private readonly Distribution DisDumping = new Beta(2, 2, 3 * 60, 7 * 60);
        #endregion

        #endregion

        #region Public Constructors

        #region ScenarioSetup()
        public ScenarioSetup()
        {
            InitializeComponent();

            theShaftAttributes.Add(shaftAttributes.State);
            theShaftAttributes.Add(shaftAttributes.CurrentDepth);
            theShaftAttributes.Add(shaftAttributes.StartTime);
            theShaftAttributes.Add(shaftAttributes.FinishTime);
            theShaftAttributes.Add(shaftAttributes.CurrentSection);
            theShaftAttributes.Add(shaftAttributes.ExcavatedDirtVolume);
            theShaftAttributes.Add(shaftAttributes.PileQuantity);
            theShaftAttributes.Add(shaftAttributes.PilingFinishTime);
            theShaftAttributes.Add(shaftAttributes.ConstructionSite);
            theShaftSectionAttributes.Add(shaftSectionAttributes.StartTime);
            theShaftSectionAttributes.Add(shaftSectionAttributes.FinishTime);
            theShaftSectionAttributes.Add(shaftSectionAttributes.State);


    theUndercutAttributes.Add(undercutAttributes.CurrentInstallationProcess);
            theUndercutAttributes.Add(undercutAttributes.CurrentSection);
            theUndercutAttributes.Add(undercutAttributes.ExcavatedDirtVolume);
            theUndercutAttributes.Add(undercutAttributes.FinishTime);
            theUndercutAttributes.Add(undercutAttributes.HasSwitch);
            theUndercutAttributes.Add(undercutAttributes.StartTime);


    theTailTunnelSectionAttributes.Add(tailTunnelSectionAttributes.FinishTime);

    theTailTunnelSectionAttributes.Add(tailTunnelSectionAttributes.LiningProdu
ctivity);

    theTailTunnelSectionAttributes.Add(tailTunnelSectionAttributes.Productivity
);

    theTailTunnelSectionAttributes.Add(tailTunnelSectionAttributes.StartTime);

    theTailTunnelSectionAttributes.Add(tailTunnelSectionAttributes.State);

            theTunnelAttributes.Add(tunnelAttributes.CrewActivity);
            theTunnelAttributes.Add(tunnelAttributes.CurrentChainage);

    theTunnelAttributes.Add(tunnelAttributes.CurrentGeotechnicalSection);
            theTunnelAttributes.Add(tunnelAttributes.ExcavatedDirtVolume);
            theTunnelAttributes.Add(tunnelAttributes.ExcavationArea);
            theTunnelAttributes.Add(tunnelAttributes.FinishTime);
            theTunnelAttributes.Add(tunnelAttributes.LastSurveyChainage);
            theTunnelAttributes.Add(tunnelAttributes.Scenario);
            theTunnelAttributes.Add(tunnelAttributes.StartTime);
            theTunnelAttributes.Add(tunnelAttributes.State);
            theTunnelAttributes.Add(tunnelAttributes.UnlinedLength);


    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.FinishTi
me);

    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.LinerTy
pe);
```

```
    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.StartTi
me);

    theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.State);

    //theGeotechnicalSectionAttributes.Add(geotechnicalSectionAttributes.NextA
dvanceRate);


    theGeometrySectionAttributes.Add(geometrySectionAttributes.FinishTime);

    theGeometrySectionAttributes.Add(geometrySectionAttributes.StartTime);

    theGeometrySectionAttributes.Add(geometrySectionAttributes.NextSection);

    theGeometrySectionAttributes.Add(geometrySectionAttributes.PreviousSecti
on);

    theGeometrySectionAttributes.Add(geometrySectionAttributes.Tunnel);

    theGeometrySectionAttributes.Add(geometrySectionAttributes.SurveyDuratio
n);

    theGeometrySectionAttributes.Add(geometrySectionAttributes.SurveyInterval
);

            theTrainAttributes.Add(trainAttributes.CapturedTrack);
            theTrainAttributes.Add(trainAttributes.CartsCounter);
            theTrainAttributes.Add(trainAttributes.CostPerHour);
            theTrainAttributes.Add(trainAttributes.CurrentDirtVolume);
            theTrainAttributes.Add(trainAttributes.EmissionFactor);
            theTrainAttributes.Add(trainAttributes.EmptySpeed);
            theTrainAttributes.Add(trainAttributes.EnergyConsumption);
            theTrainAttributes.Add(trainAttributes.EquipmentState);
            theTrainAttributes.Add(trainAttributes.LoadedSpeed);
            theTrainAttributes.Add(trainAttributes.MuckCartCount);
            theTrainAttributes.Add(trainAttributes.Position);
            theTrainAttributes.Add(trainAttributes.ResourceState);
            theTrainAttributes.Add(trainAttributes.TimeBetweenFailure);
            theTrainAttributes.Add(trainAttributes.TimeToRepair);
            theTrainAttributes.Add(trainAttributes.WorkHours);
            theTrainAttributes.Add(trainAttributes.WorkState);

            theCraneAttributes.Add(craneAttributes.EmissionFactor);
            theCraneAttributes.Add(craneAttributes.EnergyConsumption);
            theCraneAttributes.Add(craneAttributes.EquipmentState);
            theCraneAttributes.Add(craneAttributes.CostPerHour);
            theCraneAttributes.Add(craneAttributes.TimeBetweenFailure);
            theCraneAttributes.Add(craneAttributes.TimeToRepair);
            theCraneAttributes.Add(craneAttributes.WorkHours);

            theTbmAttributes.Add(tbmAttributes.EmissionFactor);
            theTbmAttributes.Add(tbmAttributes.EnergyConsumption);
            theTbmAttributes.Add(tbmAttributes.EquipmentState);
            theTbmAttributes.Add(tbmAttributes.CostPerHour);
            theTbmAttributes.Add(tbmAttributes.LiningDuration);
            theTbmAttributes.Add(tbmAttributes.PenetrationRate);
            theTbmAttributes.Add(tbmAttributes.ResettingDuration);
            theTbmAttributes.Add(tbmAttributes.ResourceState);
            theTbmAttributes.Add(tbmAttributes.WorkHours);
            theTbmAttributes.Add(tbmAttributes.WorkState);

            theScenarioAttributes.Add(scenarioAttributes.CrewCost);
            theScenarioAttributes.Add(scenarioAttributes.EquipmentCost);
            theScenarioAttributes.Add(scenarioAttributes.FinishTime);
            theScenarioAttributes.Add(scenarioAttributes.IndirectCostPerHour);
            theScenarioAttributes.Add(scenarioAttributes.MaterialCost);
            theScenarioAttributes.Add(scenarioAttributes.Name);
            theScenarioAttributes.Add(scenarioAttributes.Project);
            theScenarioAttributes.Add(scenarioAttributes.StartTime);

            theProjectAttributes.Add(projectAttributes.Name);

            this.shaftSections = new List<RectangularShaftSection>();

            this.trainDownTimes    =    new    Dictionary<ObjectInstanceHandle,
TimeKeeper>();
            this.tbmDownTimes    =    new    Dictionary<ObjectInstanceHandle,
TimeKeeper>();
            this.tbmBreakdownsTimes = new List<double>();
            this.tbmRepairTimes = new List<double>();

            // Initialize the Collectors.
            this.tbmRateCollector                    =                    new
Simphony.Modeling.Chart<DateTime>("");
            this.tbmRateCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());

            this.tbmCumRateCollector                  =                   new
Simphony.Modeling.Chart<DateTime>("");
```

```
            this.tbmCumRateCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());

            this.workingShaftProgressCollector              =               new
Simphony.Modeling.Chart<DateTime>("");
            this.workingShaftProgressCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());
            this.workingShaftProgressCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());

            this.tunnelProgressCollector              =               new
Simphony.Modeling.Chart<DateTime>("");
            this.tunnelProgressCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());
            this.tunnelProgressCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());

            this.workingShaftCostCollector              =               new
Simphony.Modeling.Chart<DateTime>("");
            this.workingShaftCostCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());
            this.workingShaftCostCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());

            this.tunnelCostCollector              =               new
Simphony.Modeling.Chart<DateTime>("");
            this.tunnelCostCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());
            this.tunnelCostCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());

            // Assume this is NOT in game mode and not started.
            this.isGame = false;
            this.isStarted = false;

            this.scenarioStartDate = default(DateTime);
            this.CurrentZeroBasedPeriod = 0;

            // Set properties of TunnelPeriodControl.
            this.tunnelPeriodControl.ScenarioSetup = this;

            // Initialize the TrainsOnline stack. Maximum of 1 and minimum of 2
trains.
            this.TrainsOnline = new Stack<Train>(2);

            // Initialize and populate the Game Tunnel sections.
            this.GAMEgeotechnicalSections = new double[]
{ this.NESTGeotechnicalSections[0],
ScenarioSetup.QueryDatabaseDouble(@"SELECT Length FROM Tunnel
WHERE Name = 'Gaming 2'") - this.NESTGeotechnicalSections[0] };
            this.GAMEgeometrySections = new double[]
{ ScenarioSetup.QueryDatabaseDouble(@"SELECT Length FROM Tunnel
WHERE Name = 'Gaming 2'") };

            #region Test Code: Schedule Generation
            // Test Code: Creates a schedule plan and adds it into the schedules
            //List<int> periodList = new List<int>();
            //List<double> progressList = new List<double>();
            //List<DateTime> dateList = new List<DateTime>();
            //for (int i = 1; i <= 3; i++)
            //{
            //    periodList.Add(i);
            //    progressList.Add(i * 5);
            //        dateList.Add(new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day, 0, 0, 0).AddDays(i * 5));
            //}
            //for (int i = 4; i <= 30; i++)
            //{
            //    periodList.Add(i);
            //    progressList.Add((i - 3) * 100);
            //        dateList.Add(new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day, 0, 0, 0).AddDays(i * 5));
            //}
            //for (int i = 0; i < 3; i++)
            //{
            //    if (i == 0)
            //    {
            //        this.workingShaftScheduleControl.DataTable.Rows[0][0] =
periodList[i];
            //        this.workingShaftScheduleControl.DataTable.Rows[0][1] =
progressList[i];
            //        this.workingShaftScheduleControl.DataTable.Rows[0][2] =
dateList[i];
            //    }
            //    else
            //    {
            //                    DataRow row =
this.workingShaftScheduleControl.DataTable.NewRow();
            //        row[0] = periodList[i];
            //        row[1] = progressList[i];
            //        row[2] = dateList[i];
            //        row[3] = progressList[i] * 100;
            //      this.workingShaftScheduleControl.DataTable.Rows.Add(row);
            //   }
            //}
            //for (int i = 3; i < 30; i++)
            //{
            //   if (i == 3)
            //   {
            //                  this.tunnelScheduleControl.DataTable.Rows[0][0]   =
periodList[i];
            //                  this.tunnelScheduleControl.DataTable.Rows[0][1]   =
progressList[i];
            //      this.tunnelScheduleControl.DataTable.Rows[0][2] = dateList[i];
            //   }
            //   else
            //   {
            //                                   DataRow      row      =
this.tunnelScheduleControl.DataTable.NewRow();
            //      row[0] = periodList[i];
            //      row[1] = progressList[i];
            //      row[2] = dateList[i];
            //      row[3] = progressList[i] * 100;
            //      this.tunnelScheduleControl.DataTable.Rows.Add(row);
            //   }
            //}
            // End Test Code
            #endregion
        }
        #endregion

        #endregion

        #region Internal Properties

        #region WorkingShaftBudget
        internal double WorkingShaftBudget
        {
            get
            {
                return this.workingShaftScheduleControl.WorkingShaftCost;
            }
        }
        #endregion

        #region TunnelBudget
        internal double TunnelBudget
        {
            get
            {
                return this.tunnelScheduleControl.TunnelCost;
            }
        }
        #endregion

        #region ProjectDuration
        internal TimeSpan ProjectDuration
        {
            get
            {
                double hours = ScenarioSetup.QueryDatabaseDouble(@"SELECT
TotalProjectDuration FROM TunnelDurations WHERE TunnelID = 'f4ac327a-
4a80-456b-8d6d-8bbd84276422'");
                return new TimeSpan((int)Math.Round(hours), 0, 0);
            }
        }
        #endregion

        #endregion

        #region Private Properties

        #region CurrentDateTime
        private DateTime CurrentDateTime
        {
            get
            {
                try
                {
                        foreach (Calendar calendar in this.calendarFactory)
                        {
                                return calendar.CurrentDateTime;
                        }
                       throw new Exception("CalendarFactory has no calendars!");
                }
                catch (Exception)
                {
                        return DateTime.Now;
                }
            }
        }
        #endregion

        #region StartDateTime
```

```
private DateTime StartDateTime
{
    get
    {
        try
        {
                foreach (Calendar calendar in this.calendarFactory)
                {
                        return calendar.StartDateTime;
                }
               throw new Exception("CalendarFactory has no calendars!");
        }
        catch (Exception)
        {
                return DateTime.Now;
        }
    }
}
#endregion

#region NextDay
private DateTime NextDay
{
    get;
    set;
}
#endregion

#region CurrentZeroBasedPeriod
private int CurrentZeroBasedPeriod
{
    get;
    set;
}
#endregion

#region PeriodShaftCost
private double PeriodShaftCost
{
    get
    {
        return this.TotalShaftCost - this.previousShaftCost;
    }
}
#endregion
```

```
#region PeriodShaftIncome
private double PeriodShaftIncome
{
    get
    {
        return this.TotalShaftIncome - this.previousShaftIncome;
    }
}
#endregion

#region TotalShaftCost
private double TotalShaftCost
{
    get;
    set;
}
#endregion

#region TotalShaftIncome
private double TotalShaftIncome
{
    get;
    set;
}
#endregion

#region PeriodTunnelCost
private double PeriodTunnelCost
{
    get
    {
        return this.TotalTunnelCost - this.previousTunnelCost;
    }
}
#endregion

#region PeriodTunnelIncome
private double PeriodTunnelIncome
{
    get
    {
        return this.TotalTunnelIncome - this.previousTunnelIncome;
    }
}
#endregion
```

```
        #region TotalTunnelCost
        private double TotalTunnelCost
        {
            get;
            set;
        }
        #endregion

        #region TotalTunnelIncome
        private double TotalTunnelIncome
        {
            get;
            set;
        }
        #endregion

        #region TrainsOnline
        private Stack<Train> TrainsOnline
        {
            get;
            set;
        }
        #endregion

        #endregion

        #region Private Federation Management Handlers

        #region fedAmb_FederationJoined
        private void fedAmb_FederationJoined(object sender, EventArgs e)
        {
            // Check whether the user wants to play the tunneling game, or simply
run a simulation.
            DialogResult result = MessageBox.Show("Would you like to run this
simulation in Game Mode?", "Game Mode", MessageBoxButtons.YesNo,
MessageBoxIcon.Question);
            if (result == DialogResult.No)
            {
                this.isGame = false;
                this.DisableGaming();
                return;
            }
            else
            {
                this.isGame = true;
            }
        }
        #endregion

        #region fedAmb_MakeInitialDeclarations
        private void fedAmb_MakeInitialDeclarations(object sender, EventArgs e)
        {
            if (this.isGame)
            {

    this.rtiAmb.PublishObjectClassAttributes(this.rtiAmb.GetObjectClassHandle(
"HLAobjectRoot.ConstructionActivity.TunnelSection.GeotechnicalSection"),
new                                                     AttributeHandleSet
{ this.geotechnicalSectionAttributes.NextAdvanceRate.GetAttributeHandle() });
            }
        }
        #endregion

        #endregion

        #region Private Object Management Handlers

        #region MyShaftGeneralFactory_RegisterInitialInstances
        private    void    MyShaftGeneralFactory_RegisterInitialInstances(object
sender, EventArgs e)
        {
            if (!this.isGame)
            {
                // Register the appropriate number of shafts.
                while (MyShaftGeneralFactory.Count < TotalRectWorkingShaft)
                {
                        theShaft                                        =
MyShaftGeneralFactory.RegisterObjectInstance();
                        var prevSection = default(ShaftSection);

                        // Register the appropriate number of shaft sections.
                        for (int count = 1; count <= TotalShaftSection; ++count)
                        {
                                var            shaftCurrentSection          =
MyShaftSectionFactory.RegisterObjectInstance();
                                //var            theShaftSection            =
MyShaftSectionFactory.RegisterObjectInstance();
                                shaftCurrentSection.NextSection             =
ObjectInstanceHandle.Empty;
                                shaftCurrentSection.PreviousSection         =
ObjectInstanceHandle.Empty;
```

```
                                shaftCurrentSection.Shaft                            =
theShaft.GetObjectInstanceHandle();

                                if (prevSection != null)
                                {
                                        prevSection.NextSection                      =
shaftCurrentSection.GetObjectInstanceHandle();
                                        shaftCurrentSection.PreviousSection          =
prevSection.GetObjectInstanceHandle();
                                }
                                else
                                {
                                        theShaft.FirstSection                        =
shaftCurrentSection.GetObjectInstanceHandle();
                                }
                                prevSection = shaftCurrentSection;
                        }
                }
            }
            else
            {
                theShaft = MyShaftGeneralFactory.RegisterObjectInstance();

                double depth = ScenarioSetup.QueryDatabaseDouble(@"SELECT
Depth FROM Tunnel WHERE Name = 'Gaming 2'");
                int sectionCount = (int)Math.Floor(depth / ShaftSectionDepth);

                for (int i = 0; i < sectionCount; i++)
                {

    this.shaftSections.Add(this.MyShaftSectionFactory.RegisterObjectInstance());
                }
            }
        }
        #endregion

        #region MyShaftGeneralFactory_InitializeInitialInstances
        private    void    MyShaftGeneralFactory_InitializeInitialInstances(object
sender, EventArgs e)
        {
            if (!this.isGame)
            {
                int Id = TotalRectWorkingShaft;

                // need to initial every shaft in the factory .For Each theShaft In
MyShaftGeneralFactory;
```

```
                foreach (RectangularShaft theShaft in MyShaftGeneralFactory)
                {
                        theShaft.State = ShaftState.Unstarted;
                        theShaft.Name = ("WorkingShaft" + Id.ToString());
                        theShaft.CurrentDepth = 0D;  //  will be allowed to change
to any.  How?????
                        theShaft.CurrentSection = ObjectInstanceHandle.Empty;
                        theShaft.HasSafetyWall = true;
                        theShaft.Shape = ShaftShape.Rectangular;
                        theShaft.PileQuantity = -1;
                        theShaft.PileWidth = 1;
                        theShaft.Purpose = ShaftPurpose.WorkingShaft;
                        theShaft.ExcavationLength = 30.29;
                        theShaft.ExcavationWidth = 6.35;
                        theShaft.Tag = Id--;
                        theShaft.FinishTime = Double.PositiveInfinity;
                        theShaft.StartTime = 0;
                        theShaft.ExcavatedDirtVolume = 0;
                        theShaft.PilingFinishTime = Double.PositiveInfinity;
                        theShaft.OnePileDrivingDuration                        =
this.onePileDrivingTime;

                        foreach          (ConstructionSite          site          in
this.MyConstructionSiteFactory)
                                theShaft.ConstructionSite                      =
site.GetObjectInstanceHandle();

                        foreach    (RectangularShaftSection    theSection    in
MyShaftSectionFactory)
                        {
                                if               (theShaft.Section.Shaft               ==
theShaft.GetObjectInstanceHandle())
                                {
                                       theSection.LinerType = LinerType.SheetPile;
                                        // theShaftSection.ShaftSectionID = 1;
                                        theSection.State                               =
ShaftSectionState.Unstarted; //ShaftSectionState.Unstarted;
                                        theSection.SoilSwellFactor = 1.35;
                                        theSection.MaterialCostPerMeter = 1575.0;
                                        theSection.MachineExcavationLength        =
29.0;
                                        theSection.MachineExcavationWidth = 6.0;
                                        theSection.Depth = 13.54D;

                                        theSection.MachineExcavationRate          =
this.shaftMachineExcavationRate;
```

```
                                        theSection.HandExcavationRate                =
this.shaftHandExcavationRate;

                                        theSection.UpdateAttributeValues();

    theSection.UnconditionalAttributeOwnershipDivestiture(theShaftSectionAttri
butes.ToArray());
                                }
                        }

                        theShaft.UpdateAttributeValues();

    theShaft.UnconditionalAttributeOwnershipDivestiture(theShaftAttributes.ToA
rray());
                }
            }
            else
            {
                int Id = TotalRectWorkingShaft;

                // need to initial every shaft in the factory .For Each theShaft In
MyShaftGeneralFactory;
                foreach (RectangularShaft theShaft in MyShaftGeneralFactory)
                {
                        theShaft.State = ShaftState.Unstarted;
                        theShaft.Name = ("WorkingShaft" + Id.ToString());
                        theShaft.CurrentDepth = 0D;  //  will be allowed to change
to any.  How?????
                        theShaft.CurrentSection = ObjectInstanceHandle.Empty;
                        theShaft.FirstSection                                   =
this.shaftSections[0].GetObjectInstanceHandle();
                        theShaft.HasSafetyWall = true;
                        theShaft.Shape = ShaftShape.Rectangular;
                        theShaft.PileQuantity = -1;
                        theShaft.PileWidth = 1;
                        theShaft.Purpose = ShaftPurpose.WorkingShaft;
                        theShaft.ExcavationLength = 30.29;
                        theShaft.ExcavationWidth = 6.35;
                        theShaft.Tag = Id--;
                        theShaft.FinishTime = Double.PositiveInfinity;
                        theShaft.StartTime = 0;
                        theShaft.ExcavatedDirtVolume = 0;
                        theShaft.PilingFinishTime = Double.PositiveInfinity;
                        theShaft.OnePileDrivingDuration = new Constant(0);
                        foreach          (ConstructionSite          site          in
this.MyConstructionSiteFactory)
                                theShaft.ConstructionSite                         =
site.GetObjectInstanceHandle();

                        for (int i = 0; i < this.shaftSections.Count; i++ )
                        {
                                this.shaftSections[i].Shaft                       =
this.theShaft.GetObjectInstanceHandle();
                                this.shaftSections[i].LinerType                   =
LinerType.SheetPile;
                                this.shaftSections[i].State                       =
ShaftSectionState.Unstarted; //ShaftSectionState.Unstarted;
                                this.shaftSections[i].SoilSwellFactor = 1.35;
                                this.shaftSections[i].MaterialCostPerMeter = 1575.0;
                                this.shaftSections[i].MachineExcavationLength     =
29.0;
                                this.shaftSections[i].MachineExcavationWidth = 6.0;

                                if (i == this.shaftSections.Count - 1)
                                {
                                        double            depth                   =
ScenarioSetup.QueryDatabaseDouble(@"SELECT    Depth    FROM    Tunnel
WHERE Name = 'Gaming 2'");
                                        this.shaftSections[i].Depth               =
ShaftSectionDepth + (depth - Math.Round(depth, 1));
                                }
                                else
                                {
                                        this.shaftSections[i].Depth               =
ShaftSectionDepth;
                                }

                                if (i > 0)
                                {
                                        this.shaftSections[i].PreviousSection     =
this.shaftSections[i - 1].GetObjectInstanceHandle();
                                }
                                else
                                {
                                        this.shaftSections[i].PreviousSection     =
ObjectInstanceHandle.Empty;
                                }

                                if (i < this.shaftSections.Count - 1)
                                {
                                        this.shaftSections[i].NextSection              =
this.shaftSections[i + 1].GetObjectInstanceHandle();
                                }
                                else
                                {
                                        this.shaftSections[i].NextSection              =
ObjectInstanceHandle.Empty;
                                }

                                this.shaftSections[i].MachineExcavationRate        =
this.shaftMachineExcavationRate;
                                this.shaftSections[i].HandExcavationRate           =
this.shaftHandExcavationRate;

                                this.shaftSections[i].UpdateAttributeValues();

    this.shaftSections[i].UnconditionalAttributeOwnershipDivestiture(theShaftSec
tionAttributes.ToArray());
                        }

                        theShaft.UpdateAttributeValues();

    theShaft.UnconditionalAttributeOwnershipDivestiture(theShaftAttributes.ToA
rray());
                }
            }
        }
        #endregion

        #region MyTunnelFactory_RegisterInitialInstances
        private void MyTunnelFactory_RegisterInitialInstances(object sender,
EventArgs e)
        {
            // Register the appropriate number of tunnels.
            while (MyTunnelFactory.Count < TotalTunnel)
            {
                this.theTunnel = MyTunnelFactory.RegisterObjectInstance();
                var prevSection = default(GeotechnicalSection);

                if (!this.isGame)
                {
                        // Register the appropriate number of geotechnical sections
for the tunnel.
                        for (int sectionNumber = 1; sectionNumber <=
NESTGeotechnicalSections.Length; ++sectionNumber)
                        {
                                var                    currentSection                  =
geotechnicalSectionFactory.RegisterObjectInstance();
                                currentSection.NextSection                             =
ObjectInstanceHandle.Empty;
                                currentSection.PreviousSection                         =
ObjectInstanceHandle.Empty;
                                currentSection.Length                                  =
NESTGeotechnicalSections[sectionNumber - 1];
                                currentSection.Tunnel                                  =
theTunnel.GetObjectInstanceHandle();
                                if (prevSection != null)
                                {
                                        prevSection.NextSection                        =
currentSection.GetObjectInstanceHandle();
                                        currentSection.PreviousSection                 =
prevSection.GetObjectInstanceHandle();
                                }
                                else
                                {
                                        theTunnel.FirstGeotechnicalSection             =
currentSection.GetObjectInstanceHandle();
                                }
                                prevSection = currentSection;
                        }

                        // Register the appropriate number of geometry sections for
the tunnel.
                        var prevGeometrySection = default(GeometrySection);
                        for  (int  sectionNumber  =  1;  sectionNumber  <=
this.NESTGeometrySections.Length; ++sectionNumber)
                        {
                                var                    currentSection                  =
MyGeometrySectionFactory.RegisterObjectInstance();
                                currentSection.NextSection                             =
ObjectInstanceHandle.Empty;
                                currentSection.PreviousSection                         =
ObjectInstanceHandle.Empty;
                                currentSection.Tunnel                                  =
theTunnel.GetObjectInstanceHandle();
                                if (prevGeometrySection != null)
                                {
                                        prevGeometrySection.NextSection                =
currentSection.GetObjectInstanceHandle();
                                        currentSection.PreviousSection                 =
prevGeometrySection.GetObjectInstanceHandle();
                                }
```

```
                else
                {
                    theTunnel.FirstGeometrySection =
currentSection.GetObjectInstanceHandle();
                }
                prevGeometrySection = currentSection;
            }
        }
        else
        {
            // Register the appropriate number of geotechnical sections
for the tunnel.
            for (int sectionNumber = 1; sectionNumber <=
this.GAMEgeotechnicalSections.Length; ++sectionNumber)
            {
                var currentSection =
geotechnicalSectionFactory.RegisterObjectInstance();
                currentSection.NextSection =
ObjectInstanceHandle.Empty;
                currentSection.PreviousSection =
ObjectInstanceHandle.Empty;
                currentSection.Length =
GAMEgeotechnicalSections[sectionNumber - 1];
                currentSection.Tunnel =
theTunnel.GetObjectInstanceHandle();
                if (prevSection != null)
                {
                    prevSection.NextSection =
currentSection.GetObjectInstanceHandle();
                    currentSection.PreviousSection =
prevSection.GetObjectInstanceHandle();
                }
                else
                {
                    theTunnel.FirstGeotechnicalSection =
currentSection.GetObjectInstanceHandle();
                }
                prevSection = currentSection;
            }

            // Register the appropriate number of geometry sections for
the tunnel.
            var prevGeometrySection = default(GeometrySection);
            for (int sectionNumber = 1; sectionNumber <=
this.GAMEgeometrySections.Length; ++sectionNumber)
            {
```

```
                                var                         currentSection                         =
MyGeometrySectionFactory.RegisterObjectInstance();
                                currentSection.NextSection                                       =
ObjectInstanceHandle.Empty;
                                currentSection.PreviousSection                                   =
ObjectInstanceHandle.Empty;
                                currentSection.Tunnel                                            =
theTunnel.GetObjectInstanceHandle();
                                if (prevGeometrySection != null)
                                {
                                        prevGeometrySection.NextSection                          =
currentSection.GetObjectInstanceHandle();
                                        currentSection.PreviousSection                           =
prevGeometrySection.GetObjectInstanceHandle();
                                }
                                else
                                {
                                        theTunnel.FirstGeometrySection                           =
currentSection.GetObjectInstanceHandle();
                                }
                                prevGeometrySection = currentSection;
                        }
                }
            }
        }
        #endregion

        #region MyTunnelFactory_InitializeInitialInstances
        private   void   MyTunnelFactory_InitializeInitialInstances(object   sender,
EventArgs e)
        {
            foreach (Tunnel theTunnel in MyTunnelFactory)
            {
                if (this.isGame)
                {
                        theTunnel.Diameter                                               =
ScenarioSetup.QueryDatabaseDouble(@"SELECT   ExcavatedDiameter   FROM
Tunnel WHERE Name = 'Gaming 2'");
                        theTunnel.Depth                                                  =
ScenarioSetup.QueryDatabaseDouble(@"SELECT    Depth    FROM    Tunnel
WHERE Name = 'Gaming 2'");
                }
                else
                {
                        theTunnel.Diameter = 2.344;
                        theTunnel.Depth = 13.54;
                }

                //theTunnel.ConveyorExtraTime = 3.5D * 60D * 60D;
                theTunnel.CrewActivity = TunnelCrewActivity.WaitingTrain;
                theTunnel.CurrentChainage = 0;
                theTunnel.CurrentGeotechnicalSection =
ObjectInstanceHandle.Empty;
                theTunnel.ExcavatedDirtVolume = 0;
                theTunnel.ExcavationArea = Math.PI * theTunnel.Diameter *
theTunnel.Diameter / 4D;
                theTunnel.ExtendTrackInterval = 5D;
                theTunnel.ExtendTrackDuration = this.DisExtendTrack;
                theTunnel.ExtendUtilityInterval = 3D;
                theTunnel.ExtendUtilityDuration = this.DisExtendUtility;
                theTunnel.FinishTime = Double.PositiveInfinity;
                theTunnel.LastSurveyChainage = 0.0;
                theTunnel.Name = "NEST";
                foreach (Scenario scenario in this.MyScenarioFactory)
                        theTunnel.Scenario = scenario.GetObjectInstanceHandle();
                theTunnel.Shape = TunnelShape.Circular;
                theTunnel.StartTime = Double.PositiveInfinity;
                theTunnel.State = TunnelState.Unstarted;
                theTunnel.SurveyInterval = 10; //every 10 meters, survey once.
                theTunnel.SurveyDuration = this.DisSurvey;
                theTunnel.SwitchInstallationChainage =
NESTGeotechnicalSections[0];
                theTunnel.UnlinedLength = 0.0;

                foreach (Undercut undercut in this.MyUndercutFactory)
                       theTunnel.Undercut = undercut.GetObjectInstanceHandle();

                foreach (GeometrySection theSection in
this.MyGeometrySectionFactory)
                {
                        if (theSection.Tunnel ==
theTunnel.GetObjectInstanceHandle())
                        {
                                theSection.FinishTime = Double.PositiveInfinity;
                                theSection.StartTime = Double.PositiveInfinity;

                                theSection.UpdateAttributeValues();

    theSection.UnconditionalAttributeOwnershipDivestiture(this.theGeometrySec
tionAttributes.ToArray());
                        }
                }
                foreach (GeotechnicalSection theSection in geotechnicalSectionFactory)
                {
                    if (theSection.Tunnel == theTunnel.GetObjectInstanceHandle())
                    {
                        theSection.State = TunnelSectionState.Unstarted;
                        theSection.SoilSwellFactor = 1.3;
                        theSection.MaterialCostPerMeter = 1575.0;
                        theSection.ExcavationMethod = TunnelExcavationMethod.TbmTunnel;

                        //if (this.isGame)
                        //{
                        //    var theValues = new AttributeHandleValueMap();
                        //    theValues.Add(this.geotechnicalSectionAttributes.NextAdvanceRate.GetAttributeHandle(), 0.5);
                        //    this.rtiAmb.UpdateAttributeValues(theSection.GetObjectInstanceHandle(), theValues, null, this.rtiAmb.QueryLogicalTime() + this.fedAmb.Lookahead);
                        //}

                        theSection.UpdateAttributeValues();

    theSection.UnconditionalAttributeOwnershipDivestiture(theGeotechnicalSectionAttributes.ToArray());
                    }
                }
                theTunnel.UpdateAttributeValues();

    theTunnel.UnconditionalAttributeOwnershipDivestiture(theTunnelAttributes.ToArray());
            }
        }
        #endregion

        #region MyUnderCutFactory_RegisterInitialInstances
        private void MyUnderCutFactory_RegisterInitialInstances(object sender, EventArgs e)
        {
            this.MyUndercutFactory.RegisterObjectInstance();
        }
        #endregion
        #region MyUnderCutFactory_InitializeInitialInstances
        private void MyUnderCutFactory_InitializeInitialInstances(object sender,
EventArgs e)
        {
            foreach (Undercut theUndercut in MyUndercutFactory)
            {
                theUndercut.CurrentInstallationProcess                    =
UndercutInstallation.None;
                theUndercut.CurrentSection = ObjectInstanceHandle.Empty;
                theUndercut.ExcavatedDirtVolume = 0.0;
                theUndercut.ExcavationArea = 0.0;
                theUndercut.FinishTime = double.PositiveInfinity;
                foreach        (TailTunnelSection        section        in
this.tailTunnelSectionFactory)
                        theUndercut.FirstSection                          =
section.GetObjectInstanceHandle();
                theUndercut.HasSwitch = false;
                theUndercut.Height = 0.0;
                theUndercut.LiningLength = 0.0;
                theUndercut.StartTime = double.PositiveInfinity;
                theUndercut.SwitchInstallationDuration = this.switchInstallTime;
                theUndercut.TailTunnel = ObjectInstanceHandle.Empty;
                theUndercut.WorkingShaft = theShaft.GetObjectInstanceHandle();

                theUndercut.UpdateAttributeValues();

    theUndercut.UnconditionalAttributeOwnershipDivestiture(theUndercutAttribu
tes.ToArray());
            }
        }
        #endregion

        #region MyTbmFactory_RegisterInitialInstances
        private   void   MyTbmFactory_RegisterInitialInstances(object   sender,
EventArgs e)
        {
            //Register the appropriate number of Tbm.
            for (int count = 1; count <= TotalTbm; ++count)
            {
                Tbm theTbm = MyTbmFactory.RegisterObjectInstance();

    theTbm.UnconditionalAttributeOwnershipDivestiture(tbmAttributes.TimeBet
weenFailure);
```

```
    theTbm.UnconditionalAttributeOwnershipDivestiture(tbmAttributes.TimeTo
Repair);

    theTbm.UnconditionalAttributeOwnershipDivestiture(tbmAttributes.TimeBet
weenMajorFailure);

    theTbm.UnconditionalAttributeOwnershipDivestiture(tbmAttributes.TimeTo
MajorRepair);

                this.tbmDownTimes.Add(theTbm.GetObjectInstanceHandle(), new
TimeKeeper());
            }
        }
        #endregion

        #region MyTbmFactory_InitializeInitialInstances
        private  void  MyTbmFactory_InitializeInitialInstances(object  sender,
EventArgs e)
        {
            foreach (Tbm theTbm in MyTbmFactory)
            {
                theTbm.AverageWaitTime = 0;
                theTbm.Diameter = 2.344;
                theTbm.EquipmentState = EquipmentState.Functional;
                theTbm.EquipmentType = EquipmentType.TBM;
                theTbm.CostPerHour = 60.0 * 60.0; //and other advance rate /
MTBF/ MTTR
                theTbm.LiningDuration = this.DisLining;
                theTbm.GantrySectionLength = 4.0;
                theTbm.GantryLength = 36.0;
                theTbm.GantryInstallationDuration = this.DisExtendGantry;
                theTbm.PenetrationRate = 1.0 / 60.0 / 60.0;
                theTbm.WorkHours = 6;
                theTbm.ResourceState = ResourceState.Idle;
                theTbm.ID = 1;
                theTbm.InstallationDuration = this.tbmInstallTime;
                theTbm.ResettingDuration = this.DisResetting;
                theTbm.ConveyorLength = 36;
                theTbm.ConveyorInstallationDuration = new Constant(4 * 60 *
60);
                theTbm.Tunnel = theTunnel.GetObjectInstanceHandle();
                theTbm.WorkState = TbmWorkState.Waiting;
                theTbm.ResourceState = ResourceState.Idle;

                theTbm.EmissionFactor = 0;
                theTbm.EnergyConsumption = 0;
                theTbm.EnergyType = EquipmentEnergyType.Electrical;

                foreach (Scenario scenario in this.MyScenarioFactory)
                        theTbm.Scenario = scenario.GetObjectInstanceHandle();

                theTbm.UpdateAttributeValues();

    theTbm.UnconditionalAttributeOwnershipDivestiture(theTbmAttributes.ToAr
ray());

                Debug.WriteLine("ScenarioSetup      is      Federate      "      +
this.fedAmb.FederateHandle);
            }
        }
        #endregion

        #region MyTrainFactory_RegisterInitialInstances
        private    void    MyTrainFactory_RegisterInitialInstances(object    sender,
EventArgs e)
        {
            //Register the appropriate number of Train.
            for (int count = 1; count <= TotalTrain; ++count)
            {
                Train theTrain = MyTrainFactory.RegisterObjectInstance();

                this.trainDownTimes.Add(theTrain.GetObjectInstanceHandle(),
new TimeKeeper());
            }
        }
        #endregion

        #region MyTrainFactory_InitializeInitialInstances
        private    void    MyTrainFactory_InitializeInitialInstances(object    sender,
EventArgs e)
        {
            int count = 1;
            foreach (Train theTrain in MyTrainFactory)
            {
                theTrain.AverageWaitTime = 0;
                theTrain.CapacityPerCart = 2; // 2 m3.
                theTrain.CapturedTrack = false;
                theTrain.CartsCounter = 0;
                theTrain.CostPerHour = 65.0;
                theTrain.CurrentDirtVolume = 0;
                theTrain.EmptySpeed = 1.5 * 1000D / 3600D;
                theTrain.EquipmentState = EquipmentState.Functional;
                theTrain.Liner = ObjectInstanceHandle.Empty;
                theTrain.LoadedSpeed = 1 * 1000D / 3600D;
                theTrain.MaterialCartCount = 1;
                theTrain.MaterialLoadingDuration = this.DisLoading;
                theTrain.MuckCartCount = TotalMuckCars;
                theTrain.MuckCartDumpDuration = this.DisDumping;
                theTrain.Position = TrainLocation.Undercut;
                theTrain.ResourceState = ResourceState.Busy;

                theTrain.TimeBetweenFailure = this.DisBreakdown;
                theTrain.TimeToRepair = this.DisRepair;
                theTrain.Tunnel = this.theTunnel.GetObjectInstanceHandle();
                theTrain.WorkState = TrainWorkState.UnStarted;

                theTrain.ID = count;
                count += 1;

                // for emission:
                theTrain.EmissionFactor = 0;
                theTrain.EnergyConsumption = 0;
                theTrain.EnergyType = EquipmentEnergyType.Electrical;
                theTrain.EquipmentType = EquipmentType.Train;
                // end of emission.

                foreach (Scenario scenario in this.MyScenarioFactory)
                    theTrain.Scenario = scenario.GetObjectInstanceHandle();

                theTrain.UpdateAttributeValues();

    theTrain.UnconditionalAttributeOwnershipDivestiture(theTrainAttributes.To
Array());
            }
        }
        #endregion

        #region myConcreteFactory_RegisterInitialInstances
        private void myConcreteFactory_RegisterInitialInstances(object sender,
EventArgs e)
        {
            myConcrete = myConcreteFactory.RegisterObjectInstance();
        }
        #endregion

        #region myConcreteFactory_InitializeInitialInstances
        private void myConcreteFactory_InitializeInitialInstances(object sender,
EventArgs e)
        {
            foreach (ConstructionSite site in this.MyConstructionSiteFactory)
                myConcrete.ConstructionSite = site.GetObjectInstanceHandle();
            myConcrete.DeliveryUsabilityProportion = new Uniform(0.9, 1.0);
            myConcrete.SectionsPerCircle = 4;
            myConcrete.OrderThreshold = 50;
            myConcrete.OrderQuantity = 100;
            myConcrete.Quantity = 100;
            myConcrete.LinerLength = 1.0;
            myConcrete.UpdateAttributeValues();

    myConcrete.UnconditionalAttributeOwnershipDivestiture(concreteAttributes.
DeliveryUsabilityProportion,                    concreteAttributes.SectionsPerCircle,
concreteAttributes.LinerLength,                          concreteAttributes.Quantity,
concreteAttributes.OrderThreshold, concreteAttributes.OrderQuantity);
        }
        #endregion

        #region MyCraneFactory_InitializeInitialInstances_1
        private void MyCraneFactory_InitializeInitialInstances_1(object sender,
EventArgs e)
        {
            foreach (Crane theCrane in MyCraneFactory)
            {
                theCrane.EquipmentState = EquipmentState.Functional;
                theCrane.WorkHours = 0;
                theCrane.CostPerHour = 223.0; // 30 in the Simphony model.
                theCrane.Load = ObjectInstanceHandle.Empty;
                theCrane.ResourceState = ResourceState.Busy;

                theCrane.TimeBetweenFailure = this.DisBreakdown;
                theCrane.TimeToRepair = this.DisRepair;

                // for emission:
                theCrane.EmissionFactor = 0;
                theCrane.EnergyConsumption = 0;
                theCrane.EnergyType = EquipmentEnergyType.Gas;
                theCrane.EquipmentType = EquipmentType.Crane;
                // end of emission.

                foreach (Scenario scenario in this.MyScenarioFactory)
                        theCrane.Scenario = scenario.GetObjectInstanceHandle();
                foreach (ConstructionSite site in this.MyConstructionSiteFactory)
```

```
                    theCrane.ConstructionSite                              =
site.GetObjectInstanceHandle();

                theCrane.UpdateAttributeValues();

    theCrane.UnconditionalAttributeOwnershipDivestiture(theCraneAttributes.To
Array());
            }
        }
        #endregion

        #region MyCraneFactory_RegisterInitialInstances_1
        private   void   MyCraneFactory_RegisterInitialInstances_1(object   sender,
EventArgs e)
        {
            //Register the appropriate number of Train.
            for (int count = 0; count < TotalRectWorkingShaft; ++count)
            {
                Crane theCrane = MyCraneFactory.RegisterObjectInstance();
            }
        }
        #endregion

        #region MyScenarioFactory_RegisterInitialInstances
        private   void   MyScenarioFactory_RegisterInitialInstances(object   sender,
EventArgs e)
        {
            this.MyScenarioFactory.RegisterObjectInstance();
        }
        #endregion

        #region MyScenarioFactory_InitializeInitialInstances
        private   void   MyScenarioFactory_InitializeInitialInstances(object   sender,
EventArgs e)
        {
            foreach (Scenario scenario in this.MyScenarioFactory)
            {
                scenario.CrewCost = 0;
                scenario.EquipmentCost = 0;
                scenario.FinishTime = double.PositiveInfinity;
                scenario.IndirectCostPerHour = 240;
                scenario.MaterialCost = 0;
                scenario.Name = "NEST scenario 1";
                foreach (Project project in this.MyProjectFactory)
                {
                        scenario.Project = project.GetObjectInstanceHandle();
                }
                scenario.StartTime = 0;
                scenario.UpdateAttributeValues();

    scenario.UnconditionalAttributeOwnershipDivestiture(theScenarioAttributes.
ToArray());
            }
        }
        #endregion

        #region MyProjectFactory_RegisterInitialInstances
        private void MyProjectFactory_RegisterInitialInstances(object sender,
EventArgs e)
        {
            this.MyProjectFactory.RegisterObjectInstance();
        }
        #endregion

        #region MyProjectFactory_InitializeInitialInstances
        private void MyProjectFactory_InitializeInitialInstances(object sender,
EventArgs e)
        {
            foreach (Project project in this.MyProjectFactory)
            {
                project.Name = "NEST";
                project.UpdateAttributeValues();

    project.UnconditionalAttributeOwnershipDivestiture(theProjectAttributes.To
Array());
            }
        }
        #endregion

        #region MyConstructionSiteFactory_RegisterInitialInstances
        private void MyConstructionSiteFactory_RegisterInitialInstances(object
sender, EventArgs e)
        {
            this.MyConstructionSiteFactory.RegisterObjectInstance();
        }
        #endregion

        #region MyConstructionSiteFactory_InitializeInitialInstances
        private void MyConstructionSiteFactory_InitializeInitialInstances(object
sender, EventArgs e)
        {
            foreach (ConstructionSite site in this.MyConstructionSiteFactory)
            {
                site.DirtVolume = 0.0;
                site.MaximumDirtVolume = double.PositiveInfinity;
                site.Name = "NEST";
                foreach (Scenario scenario in this.MyScenarioFactory)
                        site.Scenario = scenario.GetObjectInstanceHandle();

                site.UpdateAttributeValues();

    site.UnconditionalAttributeOwnershipDivestiture(constructionSiteAttributes.
DirtVolume);
            }
        }
        #endregion

        #region tailTunnelSectionFactory_RegisterInitialInstances
        private void tailTunnelSectionFactory_RegisterInitialInstances(object
sender, EventArgs e)
        {
            this.tailTunnelSectionFactory.RegisterObjectInstance();
        }
        #endregion

        #region tailTunnelSectionFactory_InitializeInitialInstances
        private void tailTunnelSectionFactory_InitializeInitialInstances(object
sender, EventArgs e)
        {
            foreach (TailTunnelSection section in this.tailTunnelSectionFactory)
            {
                section.FinishTime = double.PositiveInfinity;
                section.Length = theShaft.ExcavationLength;
                section.LiningProductivity = new Constant(double.MaxValue);
                section.MaterialCostPerMeter = 0.0; //1575.0;
                section.NextSection = ObjectInstanceHandle.Empty;
                section.PreviousSection = ObjectInstanceHandle.Empty;
                section.Productivity = new Triangular(3D / 3600D, 5D / 3600D,
4.5D / 3600D);
                section.RibSpacing = theShaft.ExcavationLength;
                section.SoilSwellFactor = 1.35;
                section.StartTime = double.PositiveInfinity;
                section.State = TailTunnelSectionState.Unstarted;
                foreach (Undercut undercut in this.MyUndercutFactory)
                        section.TailTunnel = undercut.GetObjectInstanceHandle();
                section.UpdateAttributeValues();
    section.UnconditionalAttributeOwnershipDivestiture(this.theTailTunnelSectio
nAttributes.ToArray());
            }
        }
        #endregion

        #region crewFactory_RegisterInitialInstances
        private    void    crewFactory_RegisterInitialInstances(object    sender,
EventArgs e)
        {
            this.crewFactory.RegisterObjectInstance();
        }
        #endregion

        #region crewFactory_InitializeInitialInstances
        private    void    crewFactory_InitializeInitialInstances(object    sender,
EventArgs e)
        {
            int i = 0;
            foreach (Crew crew in this.crewFactory)
            {
                crew.CostPerHour = 1;
                crew.ID = i++;
                crew.Location = new Vector3D(0, 0, 0);
                crew.NumberOfWorkers = 5;
                crew.ResourceState = ResourceState.Idle;
                crew.Scenario = this.theShaft.GetObjectInstanceHandle();

                crew.Type = "a crew";
            }
        }
        #endregion

        #endregion

        #region Private Time Management Handlers

        #region                    fedAmb_TimeAdvanceGrant(object,
TimeAdvanceGrantEventArgs)
        private    void    fedAmb_TimeAdvanceGrant(object    sender,
TimeAdvanceGrantEventArgs e)
        {
            if (!this.isGame)
            {
                this.rtiAmb.NextMessageRequest(double.MaxValue);
                return;
            }

            if (!isStarted)
            {
                // Initialize ScheduleControl.

    this.workingShaftScheduleControl.Initialize(this.CurrentDateTime);
                this.tunnelScheduleControl.Initialize(this.CurrentDateTime);

    this.retrievalShaftScheduleControl.Initialize(this.CurrentDateTime);
                this.populateScheduleButton_Click(null, null);

                // Read from database for starting scenario information.
                this.UpdateInitialProjectInfo();
                this.projectInfoControl.UpdateProjectInfo();
                this.tunnelPeriodControl.UpdateProjectInfo();
            }

            this.periodEndDate = this.CurrentDateTime;

            // Show the appropriate UI based on what is currently under
construction.
            if (this.theShaft.State != ShaftState.Finished)
            {
                this.selectCurrentActivityBox.SelectedIndex = 0;
                this.workingShaftPeriodControl.Enable();
                this.tunnelPeriodControl.Disable();
                this.retrievalShaftPeriodControl.Disable();

                this.CurrentZeroBasedPeriod                              =
this.FindNextPeriodTableIndex(this.CurrentDateTime,
this.workingShaftScheduleControl);
                this.periodNumberValueLabel.Text                         =
(this.workingShaftScheduleControl.GetPeriod(this.CurrentZeroBasedPeriod)     -
1).ToString();

                this.UpdateShaftPeriodUI();
            }
            else
            {
                this.selectCurrentActivityBox.SelectedIndex = 1;
                this.workingShaftPeriodControl.Disable();
                this.tunnelPeriodControl.Enable();
                this.retrievalShaftPeriodControl.Disable();
```

```
                    this.CurrentZeroBasedPeriod                                    =
this.FindNextPeriodTableIndex(this.CurrentDateTime,
this.tunnelScheduleControl);
                    this.periodNumberValueLabel.Text                               =
(this.tunnelScheduleControl.GetPeriod(this.CurrentZeroBasedPeriod)                 -
1).ToString();

                    this.UpdateTunnelPeriodUI();
                }

                this.UpdateProjectUI();

                // Enable the user to advance next.
                this.advanceButton.Enabled = true;

                // Send interaction informing the other federates the period is at its end.
                this.periodEndedTerminal.CurrentChainage                           =
this.theTunnel.CurrentChainage;
                this.periodEndedTerminal.FinishDate = this.CurrentDateTime;
                this.periodEndedTerminal.PeriodNumber                              =
this.CurrentZeroBasedPeriod + 1;

    this.periodEndedTerminal.SendInteraction(this.rtiAmb.QueryLogicalTime()  +
60);
            }
            #endregion

            #endregion

            #region Private Form Event Handlers

            #region advanceButton_Click(object, EventArgs)
            private void advanceButton_Click(object sender, EventArgs e)
            {
                // Check schedule settings before starting game.
                if (!isStarted)
                {
                    // Check integrities of the schedules.
                    if
(!ScheduleControl.CheckDataTableIntegrity(this.workingShaftScheduleControl.D
ataTable)
||   !ScheduleControl.CheckDataTableIntegrity(this.tunnelScheduleControl.DataTa
ble) || !this.CheckLogicalTimeFlow())
                    {
                        return;
                    }
                    // If the tables' integrity is OK, then copy the predictions to chart
collectors.

     this.PopulateCollector(this.workingShaftScheduleControl.DataTable,
this.workingShaftProgressCollector, 1, 0);
                    this.PopulateCollector(this.tunnelScheduleControl.DataTable,
this.tunnelProgressCollector, 1, 0);

     this.PopulateCollector(this.workingShaftScheduleControl.DataTable,
this.workingShaftCostCollector, 3, 0);
                    this.PopulateCollector(this.tunnelScheduleControl.DataTable,
this.tunnelCostCollector, 3, 0);

                    // Disable user input for schedules.
                    this.workingShaftScheduleControl.DisableInput();
                    this.tunnelScheduleControl.DisableInput();

                    // Synchronize the equipment selection comboboxes in other
controls to their counterparts here.
                    this.tunnelPeriodControl.TrainIDBoxSelectedIndex              =
this.trainIDBox.SelectedIndex;
                    this.tunnelPeriodControl.MuckCarIDBoxSelectedIndex            =
this.muckCarIDBox.SelectedIndex;

                    // Update initial project attributes.
                    this.UpdateInitialPeriod();
               }

               // Clear TBM's breakdown records.
               this.tbmBreakdownsTimes = new List<double>();
               this.tbmRepairTimes = new List<double>();

               // Set the parameters depending on what is currently under
construction.
               double nextPeriodLength = -1;
               if (this.theShaft.State != ShaftState.Finished)
               {
                    nextPeriodLength = this.SetNexShaftPeriod();
               }
               else
               {
                    nextPeriodLength = this.SetNextTunnelPeriod();
               }

               // Check if the ending date for next period is valid.
```

```
            if (nextPeriodLength <= 0)
            {
                MessageBox.Show("Please ensure the ending time for next period
is later than the staring time.");
                return;
            }

            // Set the scenario start time if it has not yet been set.
            if (this.scenarioStartDate == default(DateTime))
            {
                this.scenarioStartDate = this.StartDateTime;
            }

            // Advance!
            double timeNow = this.rtiAmb.QueryLogicalTime();
            this.rtiAmb.TimeAdvanceRequest(timeNow + nextPeriodLength);

            this.advanceButton.Enabled = false;
            this.isStarted = true;
        }
        #endregion

        #region ScenarioSetup_Load(object, EventArgs)
        private void ScenarioSetup_Load(object sender, EventArgs e)
        {
            this.selectActivityBox.SelectedIndex = 0;
            this.workingShaftScheduleGroup.Dock = DockStyle.Fill;
            this.tunnelScheduleGroup.Dock = DockStyle.Fill;
            this.retrievalShaftScheduleGroup.Dock = DockStyle.Fill;

            this.selectCurrentActivityBox.SelectedIndex = 0;
            this.workingShaftPeriodGroup.Dock = DockStyle.Fill;
            this.tunnelPeriodGroup.Dock = DockStyle.Fill;
            this.retrievalShaftPeriodGroup.Dock = DockStyle.Fill;
        }
        #endregion

        #region selectActivityBox_SelectedIndexChanged(object , EventArgs)
        private void selectActivityBox_SelectedIndexChanged(object sender,
EventArgs e)
        {
            switch (this.selectActivityBox.SelectedItem.ToString())
            {
                case "Working Shaft":
                        this.workingShaftScheduleGroup.Visible = true;
                        this.tunnelScheduleGroup.Visible = false;
```

```
                    this.retrievalShaftScheduleGroup.Visible = false;
                    break;
                case "Tunnel":
                    this.workingShaftScheduleGroup.Visible = false;
                    this.tunnelScheduleGroup.Visible = true;
                    this.retrievalShaftScheduleGroup.Visible = false;
                    break;
                case "Retrieval Shaft":
                    this.workingShaftScheduleGroup.Visible = false;
                    this.tunnelScheduleGroup.Visible = false;
                    this.retrievalShaftScheduleGroup.Visible = true;
                    break;
                default:
                    throw                                                    new
InvalidOperationException("this.selectActivityBox: Selection not possible!");
            }
        }
        #endregion

        #region              selectCurrentActivityBox_SelectedIndexChanged(object,
EventArgs)
        private    void    selectCurrentActivityBox_SelectedIndexChanged(object
sender, EventArgs e)
        {
            switch (this.selectCurrentActivityBox.SelectedItem.ToString())
            {
                case "Working Shaft":
                    this.workingShaftPeriodGroup.Visible = true;
                    this.tunnelPeriodGroup.Visible = false;
                    this.retrievalShaftPeriodGroup.Visible = false;
                    break;
                case "Tunnel":
                    this.workingShaftPeriodGroup.Visible = false;
                    this.tunnelPeriodGroup.Visible = true;
                    this.retrievalShaftPeriodGroup.Visible = false;
                    break;
                case "Retrieval Shaft":
                    this.workingShaftPeriodGroup.Visible = false;
                    this.tunnelPeriodGroup.Visible = false;
                    this.retrievalShaftPeriodGroup.Visible = true;
                    break;
                default:
                    throw                                                    new
InvalidOperationException("this.selectActivityBox: Selection not possible!");
            }
        }
        #endregion

        #region populateScheduleButton_Click(object , EventArgs)
        private void populateScheduleButton_Click(object sender, EventArgs e)
        {

    this.workingShaftScheduleControl.PopulateFromDatabase(ConstructionActivi
ty.WorkingShaft, 0, DateTime.Now.Date, 0);

    this.tunnelScheduleControl.PopulateFromDatabase(ConstructionActivity.Tun
nel,                              this.workingShaftScheduleControl.Periods,
this.workingShaftScheduleControl.FinishDate, 0);

    this.retrievalShaftScheduleControl.PopulateFromDatabase(ConstructionActivi
ty.RetrievalShaft,                          this.tunnelScheduleControl.Periods,
this.tunnelScheduleControl.FinishDate - new TimeSpan(2, 0, 0, 0), 0);
        }
        #endregion

        #endregion

        #region Private Message Handlers

        #region                     MyTrainFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
        private   void   MyTrainFactory_ReflectAttributeValues(object   sender,
ReflectAttributeValuesEventArgs e)
        {
            foreach (ObjectInstanceHandle train in this.trainDownTimes.Keys)
            {
                if          (e.theObject          ==          train          &&
e.theValues.Contains(this.trainAttributes.WorkState.GetAttributeHandle()))
                {
                        var                    trainState                    =
(TrainWorkState)e.theValues[this.trainAttributes.WorkState.GetAttributeHandle(
)];
                        if (trainState == TrainWorkState.WaitCrane || trainState ==
TrainWorkState.WaitTbm  ||  trainState  ==  TrainWorkState.WaitTunnelTrack
&& !this.trainDownTimes[train].IsWaiting)
                        {
                                this.trainDownTimes[train].Wait(e.theTime);
                        }
                        else  if  (trainState  !=  TrainWorkState.WaitCrane  &&
trainState    !=    TrainWorkState.WaitTunnelTrack    &&    trainState    !=
TrainWorkState.WaitTbm && this.trainDownTimes[train].IsWaiting)
                        {
```

```
                        this.trainDownTimes[train].Resume(e.theTime);
                    }
                }
            }

            if
(e.theValues.Contains(this.trainAttributes.EquipmentState.GetAttributeHandle()))
            {
                var                    equipmentState                    =
(EquipmentState)e.theValues[this.trainAttributes.EquipmentState.GetAttributeHa
ndle()];
                if (equipmentState == EquipmentState.Down)
                {
                    this.trainBreakdownCount++;
                }
            }
        }
        #endregion

        #region                    MyTbmFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
        private   void   MyTbmFactory_ReflectAttributeValues(object   sender,
ReflectAttributeValuesEventArgs e)
        {
            foreach (ObjectInstanceHandle tbm in this.tbmDownTimes.Keys)
            {
                // Check if TbmWorkState is changed.
                if        (e.theObject        ==        tbm        &&
e.theValues.Contains(this.tbmAttributes.WorkState.GetAttributeHandle()))
                {
                    var                    tbmState                    =
(TbmWorkState)e.theValues[this.tbmAttributes.WorkState.GetAttributeHandle()];
                    if      (tbmState      ==      TbmWorkState.Waiting
&& !this.tbmDownTimes[tbm].IsWaiting)
                    {
                        this.tbmDownTimes[tbm].Wait(e.theTime);
                    }
                    else   if   (tbmState   !=   TbmWorkState.Waiting   &&
this.tbmDownTimes[tbm].IsWaiting)
                    {
                        this.tbmDownTimes[tbm].Resume(e.theTime);
                    }
                }

                // Record when TBM is down and functional.
                if
(e.theValues.Contains(this.tbmAttributes.EquipmentState.GetAttributeHandle()))
                {
                        var                   equipmentState                   =
(EquipmentState)e.theValues[this.tbmAttributes.EquipmentState.GetAttributeHan
dle()];
                        if (equipmentState == EquipmentState.Down)
                        {
                                this.tbmBreakdownsTimes.Add(e.theTime);
                        }
                        else
                        {
                                this.tbmRepairTimes.Add(e.theTime);
                        }
                }
            }
        }
        #endregion

        #region                  MyTunnelFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
        private  void  MyTunnelFactory_ReflectAttributeValues(object  sender,
ReflectAttributeValuesEventArgs e)
        {
            if
(e.theValues.Contains(this.tunnelAttributes.CurrentChainage.GetAttributeHandle(
)) && !double.IsNaN(e.theTime))
            {
                DateTime                   currentDate                   =
DateTime.FromOADate(this.scenarioStartDate.ToOADate()   +   (e.theTime   /
86400));
                DateTime                  shaftFinishDate                  =
DateTime.FromOADate(this.scenarioStartDate.ToOADate()                    +
(this.theShaft.FinishTime / 86400));
                double                  currentChainage                  =
(double)e.theValues[this.tunnelAttributes.CurrentChainage.GetAttributeHandle()];

                if (this.NextDay < currentDate)
                {
                        this.NextDay = currentDate;
                }

                // If the reflect message time is after tunnel started, calculate the
TBM rate.
                if (currentDate.Subtract(this.NextDay).TotalHours >= 0)
                {
                    // Rate is calculated on a daily basis.
                    double rate = currentChainage - this.previousChainage;
                    this.tbmRateCollector.Series[0].Collect(currentDate, rate);

                    // Update the track-keeping variables.
                    this.previousChainage = currentChainage;
                    this.NextDay = this.NextDay.Add(new TimeSpan(1, 0, 0,
0));
                }

                // Record the current chainage.
                this.tunnelProgressCollector.Series[1].Collect(currentDate,
currentChainage);
            }
        }
        #endregion

        #region MyShaftGeneralFactory_ReflectAttributeValues(object,
ReflectAttributeValuesEventArgs)
        private void MyShaftGeneralFactory_ReflectAttributeValues(object
sender, ReflectAttributeValuesEventArgs e)
        {
            if
(e.theValues.Contains(this.shaftAttributes.CurrentDepth.GetAttributeHandle())
&& !double.IsNaN(e.theTime))
            {
                DateTime currentDate =
DateTime.FromOADate(this.scenarioStartDate.ToOADate() + (e.theTime /
86400));
                double currentDepth =
(double)e.theValues[this.shaftAttributes.CurrentDepth.GetAttributeHandle()];

                // Record the current depth.
                this.workingShaftProgressCollector.Series[1].Collect(currentDate,
currentDepth);
            }
        }
        #endregion

        #endregion

        #region Private Shaft-Specific Methods

        #region UpdateShaftPeriodUI()
        private void UpdateShaftPeriodUI()
        {
            this.workingShaftPeriodControl.CurrentTime = this.CurrentDateTime;
            this.workingShaftPeriodControl.ScheduleControl                    =
this.workingShaftScheduleControl;
            this.workingShaftPeriodControl.CurrentZeroBasedPeriod             =
this.CurrentZeroBasedPeriod;

            // Update the period dates.
            if (this.isStarted)
            {
                this.workingShaftPeriodControl.PeriodStartDate                =
this.periodStartDate;
                this.workingShaftPeriodControl.PeriodEndDate                  =
this.periodEndDate;
            }
            else
            {
                this.workingShaftPeriodControl.PeriodStartDate = DateTime.Now;
                this.workingShaftPeriodControl.PeriodEndDate = DateTime.Now;
            }

            // Update the current depth.
            this.workingShaftPeriodControl.CurrentDepth                       =
this.theShaft.CurrentDepth;

            // Update the excavated dirt volume.
            this.workingShaftPeriodControl.ExcavatedDirtVolume                =
this.theShaft.ExcavatedDirtVolume;

            // Update the working shaft progress.
            this.workingShaftPeriodControl.ShaftProgress                      =
Math.Round(theShaft.CurrentDepth / theShaft.GetFinishedDepth(), 4);

            // Calculate operating cost.
            if (!this.theShaft.FinishTime.IsFinite())
            {
                this.TotalShaftCost                                           +=
this.GetShaftPeriodCost(this.periodStartDate, this.periodEndDate);
            }
            else
            {
                this.TotalShaftCost                                           +=
this.GetShaftPeriodCost(this.periodStartDate,
this.StartDateTime.AddSeconds(this.theShaft.FinishTime));
            }

            // Update the period cost and ncome.
            this.TotalShaftIncome                                              =
this.workingShaftPeriodControl.ShaftProgress * this.WorkingShaftBudget;
            this.workingShaftPeriodControl.PeriodExpenditure                  =
this.PeriodShaftCost;
            this.workingShaftPeriodControl.PeriodIncome                       =
this.PeriodShaftIncome;

            // Calculate overruns
            //this.workingShaftPeriodControl.CostOverrun                      =
this.GetCostOverrun(this.previousShaftCost,             this.TotalShaftCost,
this.previousShaftProgress,                        this.theShaft.CurrentDepth,
this.workingShaftScheduleControl);
            this.workingShaftPeriodControl.CostOverrun                        =
this.GetCostOverrun(this.previousShaftCost,             this.TotalShaftCost,
this.periodStartDate, this.CurrentDateTime, this.workingShaftScheduleControl);
            this.workingShaftPeriodControl.TimeOverrun                        =
this.GetTimeOverrun(this.previousShaftProgress,     this.theShaft.CurrentDepth,
this.periodStartDate, this.periodEndDate, this.workingShaftScheduleControl);
            this.workingShaftPeriodControl.CostPerformanceIndexShaft          =
this.GetCostPerformanceIndex(this.theShaft.CurrentDepth,
this.theShaft.GetFinishedDepth(), this.WorkingShaftBudget, this.TotalShaftCost);
            this.workingShaftPeriodControl.SchedulePerformanceIndexShaft      =
this.GetSchedulePerformanceIndex(this.CurrentDateTime,
this.theShaft.CurrentDepth,                    this.theShaft.GetFinishedDepth(),
this.WorkingShaftBudget, this.workingShaftScheduleControl);

            // Collect the period cost.

    this.workingShaftCostCollector.Series[1].Collect(this.CurrentDateTime,   new
double[] { this.TotalShaftCost });

            // Set the previous period cost and income to now, after collecting.
            this.previousShaftCost = this.TotalShaftCost;
            this.previousShaftIncome = this.TotalShaftIncome;
            this.previousShaftProgress = this.theShaft.CurrentDepth;

            // Update the user control.
            this.workingShaftPeriodControl.UpdateUI();
        }
        #endregion

        #region GetShaftPeriodCost(DateTime, DateTime)
        private double GetShaftPeriodCost(DateTime periodStartDate, DateTime
periodEndDate)
        {
            if (!isStarted)
            {
                return 0;
            }

            var hours = periodEndDate.Subtract(periodStartDate).TotalHours;
            double cost = 0;
            foreach (Crane crane in this.MyCraneFactory)
            {
                cost += crane.CostPerHour * hours;
            }
            return cost;
        }
        #endregion

        #region SetNexShaftPeriod()
        private double SetNexShaftPeriod()
        {
            // Calculate difference between start and finish time of the next period.
            this.periodStartDate = this.periodEndDate;
            TimeSpan                              diff                              =
this.workingShaftPeriodControl.NextPeriodEndDate.Subtract(this.periodStartDat
e);
            return (double)diff.TotalSeconds;
        }
        #endregion

        #region ValidateShaftInput()
        private bool ValidateShaftInput()
        {
            return false;
        }
        #endregion

        #endregion

        #region Private Tunnel-Specific Methods

        #region UpdateTunnelPeriodUI()
        private void UpdateTunnelPeriodUI()
        {
            this.tunnelPeriodControl.CurrentTime = this.CurrentDateTime;
            this.tunnelPeriodControl.ScheduleControl                          =
this.tunnelScheduleControl;
            this.tunnelPeriodControl.CurrentZeroBasedPeriod                   =
this.CurrentZeroBasedPeriod;
```

```
            // Set the time which the period started and finished.
            this.tunnelPeriodControl.PeriodStartDate = this.periodStartDate;
            this.tunnelPeriodControl.PeriodEndDate = this.periodEndDate;

            // Calculate total cost of the period.
            DateTime                    shaftFinishDate                    =
DateTime.FromOADate(this.scenarioStartDate.ToOADate()                     +
(this.theShaft.FinishTime / 86400));
            if (this.periodStartDate.Subtract(shaftFinishDate).TotalDays < 0)
            {
                // If the current period starts before tunneling actually starts, add
the cost of working shaft to this period's cost.
                this.TotalShaftCost                                          +=
this.GetShaftPeriodCost(this.periodStartDate, shaftFinishDate);
                this.TotalTunnelCost                                         +=
this.GetTunnelPeriodCost(shaftFinishDate, periodEndDate);
                this.tunnelPeriodControl.PeriodExpenditure = TotalTunnelCost -
this.previousTunnelCost;

                this.UpdateShaftPeriodUI();
            }
            else
            {
                this.TotalTunnelCost                                         +=
this.GetTunnelPeriodCost(this.periodStartDate, this.periodEndDate);
                this.tunnelPeriodControl.PeriodExpenditure = TotalTunnelCost -
this.previousTunnelCost;
            }

            // Collect the period cost.
            this.tunnelCostCollector.Series[1].Collect(this.CurrentDateTime,   new
double[] { this.TotalTunnelCost });

            // Display the total excavated dirt volume and chainage & overall
progress.
            this.tunnelPeriodControl.ExcavatedDirtVolume                       =
Math.Round(this.theTunnel.ExcavatedDirtVolume, 4);
            this.tunnelPeriodControl.CurrentChainage                           =
Math.Round(this.theTunnel.CurrentChainage, 4);
            this.tunnelPeriodControl.TunnelProgress                            =
Math.Round(this.theTunnel.CurrentChainage / this.theTunnel.GetFinishedLength()
* 100, 4);
            this.TotalTunnelIncome = (this.tunnelPeriodControl.TunnelProgress *
this.TunnelBudget) + this.WorkingShaftBudget;
            this.tunnelPeriodControl.PeriodIncome  =  this.TotalTunnelIncome  -
this.previousTunnelIncome;
```

```
            this.tunnelPeriodControl.CostOverrun                                    =
this.GetCostOverrun(this.previousTunnelCost,                this.TotalTunnelCost,
this.periodStartDate, this.CurrentDateTime, this.tunnelScheduleControl);
            this.tunnelPeriodControl.TimeOverrun                                    =
this.GetTimeOverrun(this.previousTunnelProgress,
this.theTunnel.CurrentChainage,      this.periodStartDate,      this.periodEndDate,
this.tunnelScheduleControl);
            this.tunnelPeriodControl.CostPerformanceIndexTunnel                     =
this.GetCostPerformanceIndex(this.theTunnel.CurrentChainage,
this.theTunnel.GetFinishedLength(), this.TunnelBudget, this.TotalTunnelCost);
            this.tunnelPeriodControl.SchedulePerformanceIndexTunnel                 =
this.GetSchedulePerformanceIndex(this.CurrentDateTime,
this.theTunnel.CurrentChainage,              this.theTunnel.GetFinishedLength(),
this.TunnelBudget, this.tunnelScheduleControl);
            this.tunnelPeriodControl.TbmBreakdownCount                              =
this.tbmBreakdownsTimes.Count;
            this.tunnelPeriodControl.MeanTbmRepairTime = 0;
            for (int i = 0; i < this.tbmRepairTimes.Count; i++ )
            {
                if (i < this.tbmBreakdownsTimes.Count)
                {
                        this.tunnelPeriodControl.MeanTbmRepairTime              +=
this.tbmRepairTimes[i] - this.tbmBreakdownsTimes[i];
                }
            }

            this.tunnelPeriodControl.MeanTbmRepairTime                              =
Math.Round(this.tunnelPeriodControl.MeanTbmRepairTime                           /
this.tbmRepairTimes.Count / 3600);

            // Set the previous period cost and income to now, after collecting.
            this.previousTunnelCost = this.TotalTunnelCost;
            this.previousTunnelIncome = this.TotalTunnelIncome;
            this.previousTunnelProgress = this.theTunnel.CurrentChainage;

            // Calculate the average waiting time of the trains (hours per day).
            double totalDownTime = 0;
            foreach (Train train in this.MyTrainFactory)
            {
                // Updates theAverageWaitTime attribute of the current train.
                double                          hoursPerDay                         =
this.trainDownTimes[train.GetObjectInstanceHandle()].GetTotalDownTime()     /
(3600 * this.periodEndDate.Subtract(this.periodStartDate).TotalDays);
                train.AverageWaitTime = hoursPerDay;
                train.UpdateAttributeValues(this.rtiAmb.QueryLogicalTime()      +
this.fedAmb.Lookahead);
```

```
                    totalDownTime                                            +=
this.trainDownTimes[train.GetObjectInstanceHandle()].GetTotalDownTime();
                    this.trainDownTimes[train.GetObjectInstanceHandle()].Reset();
                }
                this.tunnelPeriodControl.MeanTrainWaitTime                         =
Math.Round(totalDownTime / (this.trainDownTimes.Count * 3600 *
this.periodEndDate.Subtract(this.periodStartDate).TotalDays), 4);

                // Calculate the average waiting time of the tbms (hours per day).
                totalDownTime = 0;
                foreach (Tbm tbm in this.MyTbmFactory)
                {
                    // Updates theAverageWaitTime attribute of the current tbm.
                    double                    hoursPerDay                          =
this.tbmDownTimes[tbm.GetObjectInstanceHandle()].GetTotalDownTime()       /
(3600 * this.periodEndDate.Subtract(this.periodStartDate).TotalDays);
                    tbm.AverageWaitTime = hoursPerDay;
                    tbm.UpdateAttributeValues(this.rtiAmb.QueryLogicalTime()       +
this.fedAmb.Lookahead);

                    totalDownTime                                            +=
this.tbmDownTimes[tbm.GetObjectInstanceHandle()].GetTotalDownTime();
                    this.tbmDownTimes[tbm.GetObjectInstanceHandle()].Reset();
                }
                this.tunnelPeriodControl.MeanTbmWaitTime                           =
Math.Round(totalDownTime / (this.tbmDownTimes.Count * 3600 *
this.periodEndDate.Subtract(this.periodStartDate).TotalDays), 4);

                // Calculate the total number of train breakdowns this period.
                this.tunnelPeriodControl.TrainBreakdownCount                       =
this.trainBreakdownCount;
                this.trainBreakdownCount = 0;

                // Update the periodical TBM rate chart.
                this.PopulateChart(this.tbmRateCollector,
this.tunnelPeriodControl.TbmRateChart, true);
                this.PopulateChart(this.tbmRateCollector,
this.tunnelPeriodControl.TbmCumRateChart, false);

                // Update the user control.
                this.tunnelPeriodControl.UpdateUI();
            }
            #endregion

            #region GetTunnelPeriodCost(DateTime, DateTime)
```

```
        private double GetTunnelPeriodCost(DateTime periodStartDate,
DateTime periodEndDate)
        {
            var hours = periodEndDate.Subtract(periodStartDate).TotalHours;
            double cost = 0;
            foreach (Tbm tbm in this.MyTbmFactory)
            {
                cost += tbm.CostPerHour * hours;
            }
            foreach (Crane crane in this.MyCraneFactory)
            {
                cost += crane.CostPerHour * hours;
            }
            foreach (Train train in this.MyTrainFactory)
            {
                cost += train.CostPerHour * hours;
            }
            return cost;
        }
        #endregion

        #region SetNextTunnelPeriod()
        private double SetNextTunnelPeriod()
        {
            // Update advance rate.
            this.UpdateAdvanceRate();

            // Update train attributes.
            foreach (Train train in this.MyTrainFactory)
            {
                this.tunnelPeriodControl.UpdateTrainAttributes(train,
string.Empty);
            }

            // Update muck car attributes.
            foreach (Train train in this.MyTrainFactory)
            {
                this.tunnelPeriodControl.UpdateMuckCartAttributes(train,
string.Empty);
            }

            // Clear the periodical TBM rate
            this.tbmRateCollector.Series.RemoveAt(0);
            this.tbmRateCollector.Series.Add(new
Simphony.Modeling.Series<DateTime>());
```

```
            // Calculate difference between start and finish time of the next period.
            this.periodStartDate = this.periodEndDate;
            TimeSpan                              diff                              =
this.tunnelPeriodControl.NextPeriodEndDate.Subtract(this.periodStartDate);
            return (double)diff.TotalSeconds;
        }
        #endregion

        #region ValidateTunnelInput()
        private bool ValidateTunnelInput()
        {

            return false;
        }
        #endregion

        #endregion

        #region Private Methods

        #region UpdateAdvanceRate()
        private void UpdateAdvanceRate()
        {
            // Get the distribution from database.
            double                      penetrationRateLow                        =
ScenarioSetup.QueryDatabaseDouble(@"SELECT  LowPenetrationRate  FROM
TBMs WHERE TBMNO = '" + this.tbmIDBox.SelectedValue + @"'");
            double                      penetrationRateHigh                       =
ScenarioSetup.QueryDatabaseDouble(@"SELECT  HighPenetrationRate  FROM
TBMs WHERE TBMNO = '" + this.tbmIDBox.SelectedValue + @"'");
            double                      penetrationRateMode                       =
ScenarioSetup.QueryDatabaseDouble(@"SELECT  ModePenetrationRate  FROM
TBMs WHERE TBMNO = '" + this.tbmIDBox.SelectedValue + @"'");
            var     distribution     =     new     Triangular(penetrationRateLow,
penetrationRateHigh, penetrationRateMode);

            // Transform the distribution by the crew size factor.
            distribution.Transform(this.tunnelPeriodControl.CrewSizeFactor);

            // Update each section with distribution.
            foreach          (GeotechnicalSection          section          in
this.geotechnicalSectionFactory)
            {
                var rate = ScenarioSetup.Serialize(distribution);
                var theValues = new AttributeHandleValueMap();
```

```
    theValues.Add(this.geotechnicalSectionAttributes.NextAdvanceRate.GetAttri
buteHandle(), rate);

    this.rtiAmb.UpdateAttributeValues(section.GetObjectInstanceHandle(),
theValues, null, this.rtiAmb.QueryLogicalTime() + this.fedAmb.Lookahead);
            }
        }
        #endregion

        #region UpdateTbmAttributes(Tbm)
        private void UpdateTbmAttributes(Tbm tbm)
        {
            if (!tbm.IsAttributeOwnedByFederate("Diameter"))
            {
                tbm.AttributeOwnershipAcquisition("Diameter");
            }
            if (!tbm.IsAttributeOwnedByFederate("CostPerHour"))
            {
                tbm.AttributeOwnershipAcquisition("CostPerHour");
            }
            if (!tbm.IsAttributeOwnedByFederate("TimeBetweenFailure"))
            {
                tbm.AttributeOwnershipAcquisition("TimeBetweenFailure");
            }
            if (!tbm.IsAttributeOwnedByFederate("TimeToRepair"))
            {
                tbm.AttributeOwnershipAcquisition("TimeToRepair");
            }
            if (!tbm.IsAttributeOwnedByFederate("TimeBetweenMajorFailure"))
            {
                tbm.AttributeOwnershipAcquisition("TimeBetweenMajorFailure");
            }
            if (!tbm.IsAttributeOwnedByFederate("TimeToMajorRepair"))
            {
                tbm.AttributeOwnershipAcquisition("TimeToMajorRepair");
            }

            tbm.Diameter = ScenarioSetup.QueryDatabaseDouble(@"SELECT
Diameter FROM TBMs WHERE TBMNO = '" + this.tbmIDBox.SelectedValue +
@"'");
            tbm.CostPerHour =
ScenarioSetup.QueryDatabaseDouble(@"SELECT Cost * 3600 FROM TBMs
WHERE TBMNO = '" + this.tbmIDBox.SelectedValue + @"'");
            tbm.TimeBetweenFailure =
ScenarioSetup.CalculateMeanTimeBetweenFailures(ScenarioSetup.QueryDatabas
eDouble(@"SELECT Reliability FROM TBMs WHERE TBMNO = '" +
this.tbmIDBox.SelectedValue + @"'"));
            tbm.TimeToRepair                                              =
ScenarioSetup.CalculateMeanTimeToRepair(ScenarioSetup.QueryDatabaseDoubl
e(@"SELECT Reliability FROM TBMs WHERE TBMNO = '" +
this.tbmIDBox.SelectedValue + @"'"));
            //tbm.TimeBetweenFailure = new Constant(50);
            //tbm.TimeToRepair = new Constant(10);
            tbm.TimeBetweenMajorFailure                                   =
tbm.TimeBetweenFailure.Transform(3);
            tbm.TimeToMajorRepair = tbm.TimeToRepair.Transform(2);

            tbm.UpdateAttributeValues(this.rtiAmb.QueryLogicalTime()      +
this.fedAmb.Lookahead);
        }
        #endregion

        #region UpdateCraneAttributes(Crane)
        private void UpdateCraneAttributes(Crane crane)
        {
            if (!crane.IsAttributeOwnedByFederate("CostPerHour"))
            {
                crane.AttributeOwnershipAcquisition("CostPerHour");
            }
            if (!crane.IsAttributeOwnedByFederate("TimeBetweenFailure"))
            {
                crane.AttributeOwnershipAcquisition("TimeBetweenFailure");
            }
            if (!crane.IsAttributeOwnedByFederate("TimeToRepair"))
            {
                crane.AttributeOwnershipAcquisition("TimeToRepair");
            }

            crane.CostPerHour                                             =
ScenarioSetup.QueryDatabaseDouble(@"SELECT Cost * 3600 FROM Cranes
WHERE CraneNO = '" + this.craneIDBox.SelectedValue + @"'");
            crane.TimeBetweenFailure                                      =
ScenarioSetup.CalculateMeanTimeBetweenFailures(ScenarioSetup.QueryDatabas
eDouble(@"SELECT Reliability FROM Cranes WHERE CraneNO = '" +
this.craneIDBox.SelectedValue + @"'"));
            crane.TimeToRepair                                            =
ScenarioSetup.CalculateMeanTimeToRepair(ScenarioSetup.QueryDatabaseDoubl
e(@"SELECT Reliability FROM Cranes WHERE CraneNO = '" +
this.craneIDBox.SelectedValue + @"'"));
```

```
            crane.UpdateAttributeValues(this.rtiAmb.QueryLogicalTime()          +
this.fedAmb.Lookahead);
        }
        #endregion

        #region UpdateInitialPeriod()
        private void UpdateInitialPeriod()
        {
            // Update train attributes.
            foreach (Train train in this.MyTrainFactory)
            {
                this.tunnelPeriodControl.UpdateTrainAttributes(train,
this.trainIDBox.SelectedValue.ToString());
            }

            // Update TBM attributes.
            foreach (Tbm tbm in this.MyTbmFactory)
            {
                this.UpdateTbmAttributes(tbm);
            }

            // Update Crane Attributes.
            foreach (Crane crane in this.MyCraneFactory)
            {
                this.UpdateCraneAttributes(crane);
            }

            // Update tunnel advance rate.
            this.UpdateAdvanceRate();
        }
        #endregion

        #region UpdateProjectUI()
        private void UpdateProjectUI()
        {
            // Update total project cost up until now.
            this.projectSummaryControl.TotalExpenditure = this.TotalTunnelCost
+ this.TotalShaftCost;
            this.projectSummaryControl.TotalIncome = this.TotalTunnelIncome +
this.TotalShaftIncome;
            this.projectSummaryControl.RemainingBudget = this.TunnelBudget +
this.WorkingShaftBudget - (this.TotalTunnelCost + this.TotalShaftCost);
            this.projectSummaryControl.TotalElapsedTime                        =
this.CurrentDateTime - this.StartDateTime;
            this.projectSummaryControl.RemainingTime = this.ProjectDuration -
(this.CurrentDateTime - this.StartDateTime);
```

```
            //this.projectSummaryControl.CumulativeCostOverrunShaft =
this.GetCostOverrun(0, this.TotalShaftCost, 0, this.theShaft.CurrentDepth,
this.workingShaftScheduleControl);
            this.projectSummaryControl.CumulativeCostOverrunShaft =
this.GetCostOverrun(0, this.TotalShaftCost,
this.StartDateTime.AddSeconds(this.theShaft.StartTime), this.CurrentDateTime,
this.workingShaftScheduleControl);

            if (this.theShaft.StartTime.IsFinite())
            {
                if (this.theShaft.FinishTime.IsFinite())
                {
                        this.projectSummaryControl.CumulativeTimeOverrunShaft
= this.GetTimeOverrun(0, this.theShaft.CurrentDepth, this.StartDateTime,
this.StartDateTime.AddSeconds(this.theShaft.FinishTime),
this.workingShaftScheduleControl);
                }
                else
                {
                        this.projectSummaryControl.CumulativeTimeOverrunShaft
= this.GetTimeOverrun(0, this.theShaft.CurrentDepth, this.StartDateTime,
this.CurrentDateTime, this.workingShaftScheduleControl);
                }
            }

            if (this.theTunnel.StartTime.IsFinite())
            {
                //this.projectSummaryControl.CumulativeCostOverrunTunnel =
this.GetCostOverrun(0, this.TotalTunnelCost, 0, this.theTunnel.CurrentChainage,
this.tunnelScheduleControl);
                this.projectSummaryControl.CumulativeCostOverrunTunnel =
this.GetCostOverrun(0, this.TotalTunnelCost,
this.StartDateTime.AddSeconds(this.theTunnel.StartTime), this.CurrentDateTime,
this.tunnelScheduleControl);
                this.projectSummaryControl.CumulativeTimeOverrunTunnel =
this.GetTimeOverrun(0, this.theTunnel.CurrentChainage,
this.StartDateTime.AddSeconds(this.theTunnel.StartTime), this.CurrentDateTime,
this.tunnelScheduleControl);
            }

            // Update Cost/Schdedule Performance Indices.
            this.projectSummaryControl.CostPerformanceIndexShaft =
this.GetCostPerformanceIndex(this.theShaft.CurrentDepth,
this.theShaft.GetFinishedDepth(), this.WorkingShaftBudget, this.TotalShaftCost);
```

```
            this.projectSummaryControl.CostPerformanceIndexTunnel =
this.GetCostPerformanceIndex(this.theTunnel.CurrentChainage,
this.theTunnel.GetFinishedLength(), this.TunnelBudget, this.TotalTunnelCost);
            this.projectSummaryControl.SchedulePerformanceIndexShaft =
this.GetSchedulePerformanceIndex(this.CurrentDateTime,
this.theShaft.CurrentDepth, this.theShaft.GetFinishedDepth(),
this.WorkingShaftBudget, this.workingShaftScheduleControl);
            this.projectSummaryControl.SchedulePerformanceIndexTunnel =
this.GetSchedulePerformanceIndex(this.CurrentDateTime,
this.theTunnel.CurrentChainage, this.theTunnel.GetFinishedLength(),
this.TunnelBudget, this.tunnelScheduleControl);

            // Update the progress charts.
            this.PopulateChart(this.workingShaftProgressCollector,
this.projectSummaryControl.WorkingShaftProgressChart, true);
            this.PopulateChart(this.tunnelProgressCollector,
this.projectSummaryControl.TunnelProgressChart, true);
            this.PopulateChart(this.workingShaftCostCollector,
this.projectSummaryControl.WorkingShaftCostChart, true);
            this.PopulateChart(this.tunnelCostCollector,
this.projectSummaryControl.TunnelCostChart, true);

            // Update the project summary UI.
            this.projectSummaryControl.UpdateUI();
        }
        #endregion

        #region UpdateInitialProjectInfo()
        private void UpdateInitialProjectInfo()
        {
            ScenarioSetup.PopulateComboBox(this.tbmIDBox, "TBMs",
"TBMNO", "WHERE Diameter = " + this.theTunnel.Diameter);
            ScenarioSetup.PopulateComboBox(this.trainIDBox, "Trains",
"TrainNO", "");
            ScenarioSetup.PopulateComboBox(this.muckCarIDBox, "MuckCarts",
"MuckCartNO", "");
            ScenarioSetup.PopulateComboBox(this.craneIDBox, "Cranes",
"CraneNO", "");
            ScenarioSetup.PopulateComboBox(this.excavatorIDBox, "Excavators",
"ExcavatorNO", "");
        }
        #endregion

        #endregion

        #region Private Utility Methods
        #region FindNextPeriodTableIndex()
        private int FindNextPeriodTableIndex(DateTime currentTime,
ScheduleControl control)
        {
            // Skip through all the periods whose scheduled dates the currentTime
has already passed.
            for (int i = 0; i < control.DataTable.Rows.Count; i++)
            {
                var test = control.GetDate(i);
                if (control.GetDate(i) > currentTime)
                {
                        return i;
                }
            }
            return this.CurrentZeroBasedPeriod + 1;
        }
        #endregion

        #region PopulateChart(Simphony.Modeling.Chart<DateTime>, Chart,
bool)
        private void PopulateChart(Simphony.Modeling.Chart<DateTime>
dataChart, Chart chart, bool clearPrevious)
        {
            for (int i = 0; i < dataChart.Series.Count; i++)
            {
                IEnumerable<DateTime> x;
                IEnumerable<double>[] y;

                try
                {
                        x = dataChart.Series[i].GetXValues();
                        y = dataChart.Series[i].GetYValues();
                }
                catch (InvalidOperationException)
                {
                        return;
                }

                if (clearPrevious)
                {
                        chart.Series[i].Points.Clear();
                        chart.Series[i].Points.DataBindXY(x, y);
                }
                else
                {
                    for (int j = 0; j < x.Count(); j++)
                    {
                        var xArray = x.ToArray();
                        var yArray = y.ToArray().ElementAt(0).ToArray();
                        chart.Series[i].Points.AddXY(xArray[j], yArray[j]);
                    }
                }
            }
        }
        #endregion

        #region PopulateCollector(DataTable, Chart, int)
        private void PopulateCollector(DataTable dataTable,
Simphony.Modeling.Chart<DateTime> collector, int tableYColumnIndex, int
collectorSeriesIndex)
        {
            List<DateTime> timeValues = new List<DateTime>();
            List<double> progressValues = new List<double>();

            foreach (DataRow row in dataTable.Rows)
            {
                timeValues.Add((DateTime)row[dataTable.Columns[2]]);

    progressValues.Add((double)row[dataTable.Columns[tableYColumnIndex]]);
            }

            //collector.Series.RemoveAt(0);
            //collector.Series.Add(new Simphony.Modeling.Series<DateTime>());

            for (int i = 0; i < timeValues.Count; i++)
            {
                collector.Series[collectorSeriesIndex].Collect(timeValues[i], new
double[] { progressValues[i] });
            }
        }
        #endregion

        #region CheckLogicalTimeFlow()
        private bool CheckLogicalTimeFlow()
        {
            // Ensure Tunnel and Retrieval Shaft constructions do not start until
Working Shaft is finished.
            DateTime workingShaftEndDate =
this.workingShaftScheduleControl.GetDate(this.workingShaftScheduleControl.D
ataTable.Rows.Count - 1);
            DateTime tunnelStartDate = this.tunnelScheduleControl.GetDate(0);
            if (tunnelStartDate.Subtract(workingShaftEndDate).TotalDays < 0)
            {
                MessageBox.Show("Tunnel and Retrieval Shaft construction
should not start until Working Shaft is finished.", "Schedule Timing Error");
                return false;
            }

            // Ensure the period numbers follow the above rule as well.
            int                    workingShaftEndPeriod                    =
this.workingShaftScheduleControl.GetPeriod(this.workingShaftScheduleControl.
DataTable.Rows.Count - 1);
            int tunnelStartPeriod = this.tunnelScheduleControl.GetPeriod(0);
            if (tunnelStartPeriod != workingShaftEndPeriod + 1)
            {
                MessageBox.Show("Tunnel and Retrieval Shaft construction
should not start until Working Shaft is finished.", "Schedule Timing Error");
                return false;
            }

            return true;
        }
        #endregion

        #region DisableGaming()
        private void DisableGaming()
        {
            this.ScenarioSetting.TabPages.RemoveAt(0);
            this.ScenarioSetting.TabPages.RemoveAt(0);
            this.ScenarioSetting.TabPages.RemoveAt(0);
            this.ScenarioSetting.TabPages.RemoveAt(0);
        }
        #endregion

       #region GetCostOverrun(double, double, double, double, ScheduleControl)
        [Obsolete]
        private double GetCostOverrun(double startCost, double currentCost,
double startProgress, double currentProgress, ScheduleControl control)
        {
            var currentPlannedCost = control.GetCost(currentProgress, 0);
            var startPlannedCost = control.GetCost(startProgress, 0);
            double plannedCost = currentPlannedCost - startPlannedCost;

            return ((currentCost - startCost) - plannedCost) / plannedCost * 100;
        }
        #endregion
        #region GetCostOverrun(double, double, DateTime, DateTime, ScheduleControl)
        private double GetCostOverrun(double startCost, double currentCost, DateTime startDate, DateTime currentDate, ScheduleControl control)
        {
            var currentPlannedCost = control.GetCost(currentDate, 0);
            var startPlannedCost = control.GetCost(startDate, 0);
            double plannedCost = currentPlannedCost - startPlannedCost;

            return ((currentCost - startCost) - plannedCost) / plannedCost * 100;
        }
        #endregion

        #region GetTimeOverrun(double, double, DateTime, DateTime, ScheduleControl)
        //private double GetTimeOverrun(double startProgress, double currentProgress, DateTime startDate, DateTime currentDate, ScheduleControl control)
        //{
        //    DateTime plannedFinishDate = control.GetDate(currentProgress);

        //    var firstTime = startProgress;
        //    var secondTime = currentProgress;

        //    var firstValue = (startDate - this.StartDateTime).TotalDays;
        //    var secondValue = (plannedFinishDate - this.StartDateTime).TotalDays;

        //    var t = (currentDate - this.StartDateTime).TotalDays;
        //     double plannedLength = ScenarioSetup.LinearInterpolation(firstTime, firstValue, secondTime, secondValue, t);
        //    double actualLength = (currentDate - startDate).TotalDays;

        //    return (actualLength - plannedLength) / plannedLength * 100;
        //}

        //private double GetTimeOverrun(double startProgress, double currentProgress, DateTime startDate, DateTime currentDate, ScheduleControl control)
        //{
        //    var actualDuration = (currentDate - startDate).TotalDays;

        //    var plannedStartDate = control.GetDate(startProgress);
        //    var plannedCurrentDate = control.GetDate(currentProgress);
        //    var plannedDuration = (plannedCurrentDate - plannedStartDate).TotalDays;
        //    return (actualDuration - plannedDuration) / plannedDuration * 100;
        //}

        private double GetTimeOverrun(double startProgress, double
currentProgress, DateTime startDate, DateTime currentDate, ScheduleControl
control)
        {
            var actualDuration = (currentDate - startDate).TotalDays;

            var plannedStartDate = control.GetInterpolatedDate(startProgress);
            var plannedCurrentDate =
control.GetInterpolatedDate(currentProgress);
            var plannedDuration = (plannedCurrentDate -
plannedStartDate).TotalDays;

            return (actualDuration - plannedDuration) / plannedDuration * 100;
        }
        #endregion

        #region GetCostOverrun(double, DateTime, ScheduleControl)
        //[Obsolete]
        //private double GetCostOverrun(double actual, DateTime date,
ScheduleControl control)
        //{
        //    if (control.FindRowIndex(date) == -1)
        //    {
        //        return -1;
        //    }

        //    int secondIndex = control.FindRowIndex(date);
        //    int firstIndex = Math.Max(0, secondIndex - 1);

        //    double firstTime = (control.GetDate(firstIndex, DateTime.MaxValue)
- this.StartDateTime).TotalDays;
        //    double secondTime = (control.GetDate(secondIndex,
DateTime.MaxValue) - this.StartDateTime).TotalDays;

        //    double firstValue = control.GetCost(firstIndex, 0);
        //    double secondValue = control.GetCost(secondIndex, 0);

        //    double timeNow = (date - this.StartDateTime).TotalDays;

        //    double planned = LinearInterpolation(firstTime, firstValue,
secondTime, secondValue, timeNow);
```

```
        //    return (actual - planned) / planned * 100;
        //}
        #endregion

        #region GetCostPerformanceIndex(double, double, double, double)
        private double GetCostPerformanceIndex(double currentProgress, double
totalProgress, double totalBudget, double totalCost)
        {
            double earnings = totalBudget * (currentProgress / totalProgress);
            return earnings / totalCost;
        }
        #endregion

        #region GetSchedulePerformanceIndex(double, double, double, double)
        private double GetSchedulePerformanceIndex(DateTime currentDate,
double currentProgress, double totalProgress, double totalBudget,
ScheduleControl control)
        {
            double actualEarnings = totalBudget * (currentProgress /
totalProgress);
            double plannedProgress =
control.GetInterpolatedProgress(currentDate);
            double plannedEarnings = totalBudget * (plannedProgress /
totalProgress);

            return actualEarnings / plannedEarnings;
        }
        #endregion

        #endregion

        #region Public Static Utility Methods

        #region LinearInterpolation(double, double, double, double, double)
        public static double LinearInterpolation(double time1, double value1,
double time2, double value2, double t)
        {
            double value = value1 + (value2 - value1) * (t - time1) / (time2 -
time1);
            return value;
        }
        #endregion

        #region CalculateMeanTimeBetweenFailures(double)
        public static Distribution CalculateMeanTimeBetweenFailures(double
reliability)
```

```
{
    reliability.ExceptionIfOutOfRange(0, 100, "reliability");

    double low = -1;
    double medium = -1;
    double high = -1;

    if (reliability >= 0 && reliability < 50)
    {
        low = 450;
        medium = 600;
        high = 750;
    }
    else if (reliability >= 50 && reliability < 60)
    {
        low = 850;
        medium = 1000;
        high = 1150;
    }
    else if (reliability >= 60 && reliability < 70)
    {
        low = 1000;
        medium = 1150;
        high = 1300;
    }
    else if (reliability >= 70 && reliability < 80)
    {
        low = 2000;
        medium = 2200;
        high = 3000;
    }
    else if (reliability >= 80 && reliability < 90)
    {
        low = 2600;
        medium = 2750;
        high = 3100;
    }
    else if (reliability >= 90 && reliability < 1000)
    {
        low = 3300;
        medium = 3650;
        high = 4050;
    }
    else
    {
        throw new ArgumentOutOfRangeException("reliability");
    }

    return new Triangular(low, high, medium);
    //return new Triangular(low, high, medium).Transform(3600);
}
#endregion

#region CalculateMeanTimeToRepair(double)
public static Distribution CalculateMeanTimeToRepair(double reliability)
{
    reliability.ExceptionIfOutOfRange(0, 100, "reliability");

    double low = -1;
    double medium = -1;
    double high = -1;

    if (reliability >= 0 && reliability < 50)
    {
        low = 14;
        medium = 19.5;
        high = 26.5;
    }
    else if (reliability >= 50 && reliability < 60)
    {
        low = 13;
        medium = 18;
        high = 24;
    }
    else if (reliability >= 60 && reliability < 70)
    {
        low = 9;
        medium = 12;
        high = 20;
    }
    else if (reliability >= 70 && reliability < 80)
    {
        low = 5;
        medium = 15;
        high = 20;
    }
    else if (reliability >= 80 && reliability < 90)
    {
        low = 3;
        medium = 9;
        high = 16;
    }
```

```
            else if (reliability >= 90 && reliability < 1000)
            {
                low = 1;
                medium = 5;
                high = 9;
            }
            else
            {
                throw new ArgumentOutOfRangeException("reliability");
            }

            return new Triangular(low, high, medium);
            //return new Triangular(low, high, medium).Transform(3600);
        }
        #endregion

        #region FindEarlierDate(DateTime, DateTime)
        public static DateTime FindEarlierDate(DateTime date1, DateTime date2)
        {
            if (date1 <= date2)
            {
                return date1;
            }
            else
            {
                return date2;
            }
        }
        #endregion

        #region FormatTimeSpan(TimeSpan)
        public static string FormatTimeSpan(TimeSpan span)
        {
            double days = span.TotalSeconds / (3600 * 24);
            double weeks = Math.Floor(days / 7);
            days = days % 7;

            return weeks + " weeks, " + Math.Round(days) + " days"; ;
        }
        #endregion

        #region PopulateComboBox(ComboBox, string, string, string)
        public static void PopulateComboBox(ComboBox comboBox, string
tableName, string columnName, string additional)
        {
            tableName.ExceptionIfNullOrEmpty("tableName");
            columnName.ExceptionIfNullOrEmpty("columnName");

            try
            {
                // Open an SQL connection.
                SqlConnection connection = new SqlConnection(@"Data
Source=SQL\SQL2008;Initial Catalog=Tunneling;Integrated Security=True");
                connection.Open();
                SqlDataAdapter dataAdapter = new SqlDataAdapter(@"select " +
@columnName + @" from " + @tableName + " " + @additional, connection);

                // Fill the data table.
                DataTable table = new DataTable();
                dataAdapter.Fill(table);

                // Populate the data grid view.
                BindingSource bindingSource = new BindingSource();
                bindingSource.DataSource = table;
                comboBox.DataSource = bindingSource;
                comboBox.DisplayMember = columnName;
                comboBox.ValueMember = columnName;

                // Optional: set default selected index to 0.
                comboBox.SelectedIndex = 0;

                // Close the SQL connection.
                connection.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        #endregion

        #region PopulateLabel(Label, string)
        public static void PopulateLabel(Label label, string query)
        {
            query.ExceptionIfNullOrEmpty("query");

            try
            {
                // Open an SQL connection.
                SqlConnection connection = new SqlConnection(@"Data
Source=SQL\SQL2008;Initial Catalog=Tunneling;Integrated Security=True");
                connection.Open();
                SqlDataAdapter dataAdapter = new SqlDataAdapter(query,
connection);

                // Fill the data table.
                DataTable table = new DataTable();
                dataAdapter.Fill(table);

                // Populate the data grid view.
                BindingSource bindingSource = new BindingSource();
                bindingSource.DataSource = table;

                label.Text = (table.Rows[0][0]).ToString();

                // Close the SQL connection.
                connection.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        #endregion

        #region PopulateDataGridView(DataGridView, string)
        public static void PopulateDataGridView(DataGridView dataGridView,
string query)
        {
            query.ExceptionIfNullOrEmpty("query");

            try
            {
                // Open an SQL connection.
                SqlConnection connection = new SqlConnection(@"Data
Source=SQL\SQL2008;Initial Catalog=Tunneling;Integrated Security=True");
                connection.Open();
                SqlDataAdapter dataAdapter = new SqlDataAdapter(query,
connection);

                // Fill the data table.
                DataTable table = new DataTable();
                dataAdapter.Fill(table);

                // Populate the data grid view.
                BindingSource bindingSource = new BindingSource();
                bindingSource.DataSource = table;
                dataGridView.DataSource = bindingSource;
```

```
                    // Close the SQL connection.
                    connection.Close();

                    // Optional: Format the grid headers with units.
                   foreach (DataGridViewColumn column in dataGridView.Columns)
                    {
                            switch (column.HeaderText)
                            {
                                    case ("Cost"):
                                            column.HeaderText = column.HeaderText +
" ($/hr)";
                                            break;
                                    case ("Reliability"):
                                            column.HeaderText = column.HeaderText +
" (%)";
                                            break;
                                    case ("BucketSize"):
                                            column.HeaderText = column.HeaderText +
" (m³)";
                                            break;
                                    case ("PenetrationRate"):
                                            column.HeaderText = column.HeaderText +
" (Low, High, Mode); (m/hr)";
                                            break;
                                    case ("ProductionRate"):
                                            column.HeaderText = column.HeaderText +
" (m³/hr)";
                                            break;
                                    case ("Capacity"):
                                            column.HeaderText = column.HeaderText +
" (m³)";
                                            break;
                                    case ("Diameter"):
                                            column.HeaderText = column.HeaderText +
" (m)";
                                            break;
                                    case ("StrokeLength"):
                                            column.HeaderText = column.HeaderText +
" (m)";
                                            break;
                                    case ("GantryLength"):
                                            column.HeaderText = column.HeaderText +
" (m)";
                                            break;
                                    case ("SpeedEmpty"):
                                            column.HeaderText = column.HeaderText +
" (m/s)";
                                            break;
                                    case ("SpeedLoaded"):
                                            column.HeaderText = column.HeaderText +
" (m/s)";
                                            break;
                            }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
            #endregion

            #region QueryDatabaseDouble(string)
            public static double QueryDatabaseDouble(string query)
            {
                // Return the object.
                return Convert.ToDo
uble(ScenarioSetup.QueryDatabaseObject(query));
            }
            #endregion

            #region QueryDatabaseInt(string)
            public static int QueryDatabaseInt(string query)
            {
                // Return the object.
                return Convert.ToInt32(ScenarioSetup.QueryDatabaseObject(query));
            }
            #endregion

            #region QueryDatabaseObject(string query)
            public static object QueryDatabaseObject(string query)
            {
                query.ExceptionIfNullOrEmpty("tableName");

                try
                {
                    // Open an SQL connection.
                    SqlConnection    connection    =    new    SqlConnection(@"Data
Source=SQL\SQL2008;Initial Catalog=Tunneling;Integrated Security=True");
                    connection.Open();
                    SqlDataAdapter   dataAdapter   =   new   SqlDataAdapter(query,
connection);

                    // Fill the data table.
                    DataTable table = new DataTable();
                    dataAdapter.Fill(table);

                    // Close the SQL connection.
                    connection.Close();

                    // Return the object.
                    return table.Rows[0][0];
                }
                catch (Exception ex)
                {
                    MessageBox.Show("ScenarioSetup.QueryDatabase:" + ex);
                    return null;
                }
            }
            #endregion

            #region Serialize()
            /// <summary>
            /// Serializes the given object so it may be reflectable in another federate
using Cosye.Framework.
            /// </summary>
            /// <param name="theValue">The object to be serialized.</param>
            /// <returns>The byte array containing the serialized object.</returns>
            public static byte[] Serialize(object theValue)
            {
                BinaryFormatter formatter = new BinaryFormatter();
                MemoryStream stream = new MemoryStream();
                formatter.Serialize(stream, theValue);
                return stream.ToArray();
            }
            #endregion

            #endregion
        }
}
```

Appendix B. Look-ahead schedule/plan in Last Planner® System

The Last Planner® System (LPS) was developed by the staff of the Lean Construction Institute in the United States. The Last Planner™ was first explicitly defined in publication by Ballard (1994). The LPS was proposed as a control system by Ballard (2000). The implementation of LPS has been increased remarkably after the introduction by Ballard and Howell (Ballard and Howell 1998; 2003).

LPS focuses on planning reliability through project execution. LPS requires reliable plans made by last planners who can collaboratively work together and promise the completion according to these plans. Traditional planning process, which is based on the WBS (work breakdown structure) of a final product and produces a schedule indicating what "should" be done, was criticized for ignoring work logic of producing the product (Ballard and Howell 2003). On the contrary, LPS takes resources and constraints into account in planning, which reflects what "can" be done. What "should" be done is covered in a master schedule. When adding what "can" be done into what "should" be done, a detailed near-term plan of what "will" be done can be established (Ballard 2000). Consequently, the plan is achieved because of high reliability, and turns out to be what was "done."

LPS emphasizes making reasonable and achievable plans continuously, so as to ensure that the overall project goal is achieved, step-by-step. LPS tries to improve predictability of project workflow through collaborative production planning and collaborative program coordination, utilizing and activating the network of commitments. A project becomes more predictable through execution, as described in reliable plans. These plans are made through a series of planning conversations of the key project performers – the last planners, (such as the foremen on site, supplier team leaders), and project managers (Mossman 2008). The main conversation which assures the successful implementation of LPS embraces identifying work needed to achieve milestones, making that work ready, assuring a responsible individual has promised to complete it, and evaluating and learning from experience and practice.

Additionally, when applying LPS to a real project, the planning reliability is measured using percent plan complete (PPC). The collected project progress can be measured against the weekly plans, and then PPC is calculated, which can help reasoning failures and low production. This is not a part of the function provided in this research, and is not discussed. If the scheduled work is not completed as planned, the corresponding last planner(s) will take the responsibility according to the reports.

References

Abourizk, S. (2006). "Collaborative Simulation Framework for Multi-user Decision Support in Construction."

Abourizk, S. (2010). "Role of simulation in construction engineering and management." *Journal of Construction Engineering and Management*, 136(10), 1140-1153.

AbouRizk, S. M., Hague, S., Al-Bataineh, M., and Fernando, S. (2009). "Collaborative tunneling simulation using synthetic environments." *Proc., 2009 Construction Research Congress - Building a Sustainable Future, April 5, 2009 - April 7, 2009*, American Society of Civil Engineers, 1223-1232.

AbouRizk, S. M., and Hajjar, D. (1998). "A framework for applying simulation in construction." *Canadian Journal of Civil Engineering*, 25(3), 604-617.

Al-Bataineh, M. (2008). "Scenario-based planning for tunnelling construction." Ph.D. Ph.D. thesis, University of Alberta, Edmonton, ALberta, Canada.

Al-Tabtabai, H., Kartam, N., Flood, I., and Alex, A. P. (1997). "Construction project control using artificial neural networks." *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, 11(1), 45-57.

Alarcon, L. F., and Calderon, R. "Implementing Lean Production Strategies in Construction Companies." *Proc., Construction Research Congress, Winds* *of Change: Integration and Innovation in Construction, Proceedings of the Congress, March 19, 2003 - March 21*, American Society of Civil Engineers, 313-320.

Altshuler, A. A., and Luberoff, D. E. (2003). *Mega-Projects: The Changing Politics of Urban Public Investment*, Brookings Institution Press

Ballard, G. "The Last Planner." *Proc., Northern California Construction Institute Spring Conference*, Lean Construction Institute.

Ballard, G. (2000). "The Last Planner System of Production Control." DOCTOR OF PHILOSOPHY, University of Birmingham.

Ballard, G., and Howell, G. (1998). "Shielding Production: Essential Step in Production Control." *Journal of Construction Engineering and Management*, 124(1), 11-17.

Ballard, G., and Howell, G. "An Update on Last Planner." *Proc., Proceedings of the 11th Annual Meeting of the International Group for Lean Construction*, Lean Construction Institute.

Bengtsson, N., Shao, G., Johansson, B., Lee, Y. T., Leong, S., Skoogh, A., and McLean, C. "Input Data Management Methodology for Discrete Event Simulation." *Proc., 2009 Winter Simulation Conference*, IEEE, 1335-1344.

Boulonne, A., Johansson, B., Skoogh, A., and Aufenanger, M. "Simulation data architecture for sustainable development." *Proc., 2010 43rd Winter*

Simulation Conference, WSC'10, December 5, 2010 - December 8, 2010, Institute of Electrical and Electronics Engineers Inc., 3435-3446.

Chan, C. T. W., and Sin, H. C. "A web-based self-tuning control system to automate materials procurement in large construction projects." *Proc., ICCAS-SICE 2009 - ICROS-SICE International Joint Conference 2009, August 18, 2009 - August 21*, Society of Instrument and Control Engineers (SICE), 2068-2073.

Chan, S.-L., and Leung, N.-N. (2004). "Prototype web-based construction project management system." *Journal of Construction Engineering and Management*, 130(6), 935-943.

Charoenngam, C., Coquinco, S. T., and Hadikusumo, B. H. W. (2003). "Web-based application for managing change orders in construction projects." *Construction Innovation (Sage Publications, Ltd.)*, 3(4), 197-215.

Chou, J.-S., and Chong, W. K. "A web-based framework of project performance and control system." *Proc., 2008 IEEE International Conference on Robotics, Automation and Mechatronics, RAM 2008, September 21,2008 - September 24*, Inst. of Elec. and Elec. Eng. Computer Society, 803-807.

Chung, T. H., Mohamed, Y., and Abourizk, S. (2006). "Bayesian updating application into simulation in the North Edmonton Sanitary Trunk tunnel project." *Journal of Construction Engineering and Management*, 132(8), 882-894.

Defense, M., and Simulation, O. (1996). "HLA Time Management Design Document." <http://www.cc.gatech.edu/computing/pads/PAPERS/HLA-TM-1.0.pdf>. (March 30, 2011, 2010).

Diekmann, J. E., and Al-Tabtabai, H. (1992). "Knowledge-based approach to construction project control." *International Journal of Project Management*, 10(1), 23-30.

Edwards, C. (2009). "Government Cost Overruns." Cato Institute.

Fleming, Q. W., and Koffleman, J. M. (2010). *Earned Value Project Management*, Project Management Institute.

Flyvbjerg, B. (2007). "Policy and planning for large-infrastructure projects: Problems, causes, cures." *Environment and Planning B: Planning and Design*, 34(GEOBASE), 578-597.

Flyvbjerg, B., Holm, M. S., and Buhl, S. (2002). "Underestimating costs in Public Works Projects: Error or Lie?" *Journal of the American Planning Association*, 68(GEOBASE), 279-296.

Ghanem, A. G., and Abdelrazig, Y. A. "A framework for real-time construction project progress tracking." *Proc., Earth and Space 2006 - 10th Biennial International Conference on Engineering, Construction, and Operations in Challenging Environments, March 5, 2006 - March 8*, American Society of Civil Engineers, 112.

Gonzalez, V., Alarcon, L. F., Maturana, S., Mundaca, F., and Bustamante, J. (2010). "Improving planning reliability and project performance using the reliable commitment model." *Journal of Construction Engineering and Management*, 136(10), 1129-1139.

Gonzalez, V., Alarcon, L. F., and Mundaca, F. (2008). "Investigating the relationship between planning reliability and project performance." *Production Planning and Control*, 19(5), 461-474.

Government Accountability Office (2003). "Federal Aid Highways: Cost and Oversight of Major Highway and Bridge Projects: Issues and Options." GAO-03-764T.

Halpin, D. W. (1977). "CYCLONE - method for modeling job site processes." *American Society of Civil Engineers, Journal of the Construction Division*, 103(3), 489-499.

Hanna, A. S. (2011). "Using the Earned Value Management System to Improve Electrical Project Control." *Journal of Construction Engineering and Management*, 1(1), 307.

Hinze, J. (1998). *Construction planning and scheduling*, Prentice Hall, Upper Saddle River, NJ.

Hwang, S., and Liu, L. Y. "Proactive project control using productivity data and time series analysis." *Proc., 2005 ASCE International Conference on*

Computing in Civil Engineering, July 12,2005 - July 15, American Society of Civil Engineers, 1925-1935.

Ibbs, C. W., Wong, C. K., and Kwak, Y. H. (2001). "Project Change Management System." *Journal of Management in Engineering*, 17(3), 159-165.

IEEE Computer Society (2010). "IEEE Standard for Modeling and Simulation (M&amp;S) High Level Architecture (HLA)-- Framework and Rules." *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, 1-38.

Jia, G., Xue, X., Cao, L., and Chen, J. "Integrated schedule information model for airport construction projects." *Proc., International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2008, December 19, 2008 - December 21*, Inst. of Elec. and Elec. Eng. Computer Society, 71-77.

Kartam, N. A. (1996). "Making Effective Use of Construction Lessons Learned in Project Life Cycle." *Journal of Construction Engineering and Management*, 122(1), 14-21.

Kim, Y.-W., and Ballard, G. (2010). "Management thinking in the earned value method system and the last planner system." *Journal of Management in Engineering*, 26(4), 223-228.

Kuhl, F., Dahmann, J., and Weatherly, R. (2000). *Creating computer simulation systems : an introduction to the high level architecture*, Prentice Hall PTR, Upper Saddle River, NJ.

Lee, S., Pena-Mora, F., and Park, M. (2006). "Web-Enabled System Dynamics Model for Error and Change Management on Concurrent Design and Construction Projects." *Journal of Computing in Civil Engineering*, 20(4), 290-300.

Lewis, R., and Murphy, S. (2003). ""Easy Pass" series of reports." *The Boston Globe*.

Li, J., Moselhi, O., and Alkass, S. (2006). "Internet-based database management system for project control." *Engineering, Construction and Architectural Management*, 13(3), 242-253.

Li, L. (2003). "DSP hardware surrogate for the HLA." M.A.Sc., Carleton University (Canada), Canada.

Likhitruangsilp, V., and Ioannou, P. G. "Stochastic Evaluation of Tunneling Performance Using Discrete-Event Simulation." ASCE, 109.

Lu, K. (2006). "Data distribution management schemes for HLA-compliant distributed simulation systems." M.C.S., University of Ottawa, Edmonton, Alberta, Canada.

Macomber, H. (2004). “Look-ahead planning.” *Reforming project management,* <http://www. reformingprojectmanagement.com/2004/12/16/436/> (April, 2012).

Mantel, S. J., Meredith, J. R., Shafer, S. M., and Sutton, M. M. (2010). *Construction planning and scheduling*, John Wiley & Sons, Upper Saddle River, NJ.

Marshall, R. (2007). "The contribution of earned value management to project success on contracted efforts: A quantitative statistics approach within the population of experienced practitioners." *Journal of Contract Management*, Summer, 21-33.

Martinez, J. C., and Ioannou, P. G. "General purpose simulation with Stroboscope." *Proc., Proceedings of Winter Simulation Conference, 11-14 Dec. 1994*, IEEE, 1159-1166.

Marzouk, M., Abdallah, M., and El-Said, M. (2010). "Modeling Microtunneling Projects using Computer Simulation." *Journal of Construction Engineering and Management*, 136(6), 670-682.

Memon, Z. A., Abd.Majid, M. Z., and Mustaffar, M. "An automatic project progress monintoring model by integrating auto CAD and digital photos." *Proc., 2005 ASCE International Conference on Computing in Civil Engineering, July 12, 2005 - July 15*, American Society of Civil Engineers, 1605-1617.

Mitropoulos, P., and Nichita, T. (2010). "Critical Concerns of Production Control System on Projects with Labor Constraints: Lessons from a Residential Case Study." *Journal of Management in Engineering*, 26(3), 153-159.

Molenaar, K., Anderson, S., and Schexnayder, C. (2009). "Risk Analysis Tools and Management Practices to Control Transportation Project Costs." Transportation Research Board, Washington, D.C.

Molenaar, K. R. (2005). "Programmatic cost risk analysis for highway megaprojects." *Journal of Construction Engineering and Management*, 131(Compendex), 343-353.

Moon, Y. B., and Phatak, D. (2005). "Enhancing ERP system's functionality with discrete event simulation." *Industrial Management and Data Systems*, 105(9), 1206-1224.

Moselhi, O., Li, J., and Alkass, S. (2004). "Web-based integrated project control system." *Construction Management and Economics*, 22(1), 35-46.

Mossman, A. (2008). "Last Planner 5 crucial conversations for reliable flow and project delivery."

Naoum, S. G. (1994). "Critical Analysis of Time and Cost of Management and Traditional Contracts." *Journal of Construction Engineering and Management*, 120(4), 687-705.

Narayan, S. J. L. (2011). "Time & Cost Overruns in Implementation of Infrastructure Projects, Problems and Remedies." <http://www.nbmcw.com/articles/project-management-arbitration/20898-time-a-cost-overruns-in-implementation-of-infrastructure-projects.html>. (August 15, 2011).

Nassar, N. K. (2005). "An integrated framework for evaluation, forecasting and optimization of performance of construction projects." Ph.D., University of Alberta (Canada), Canada.

Navon, R. (2005). "Automated project performance control of construction projects." *Automation in Construction*, 14(4), 467-476.

Nieto-Morote, A., and Ruz-Vila, F. (2011). "Last Planner Control System Applied to a Chemical Plant Construction." *Journal of Construction Engineering and Management*, 1(1), 298.

NIST/SEMATECH (2003). *e-Handbook of Statistical Methods*, Boca Raton, Flor.

Olumide, A. O., Anderson, S. D., and Molenaar, K. R. (2010). "Sliding-scale contingency for project development process." *Transportation Research Record*(Compendex), 21-27.

Ourdev, I., Abourizk, S., and Al-Bataineh, M. "Simulation and uncertainty modeling of project schedules estimates." *Proc., 2007 Winter Simulation Conference, 9-12 Dec. 2007*, IEEE, 2128-2133.

Ourdev, I., Xie, H., and AbouRizk, S. (2008). "An Intelligent Agent Approach to Adaptive Project Management." *Tsinghua Science and Technology*, 13(Journal Article), 121-125.

Park, J. (2005). "A framework to model complex systems via distributed simulation: A case study of the virtual test bed simulation system using the high level architecture." Ph.D., University of Central Florida, Florida, United States.

Pidd, M. (1998). *Computer simulation in management science*, John Wiley & Sons, Inc.

Project Management Institute (2004). *A guide to the project management body of knowledge (PMBOK guide)*, Project Management Institute, Inc., Newtown Square, Pa.

Project Management Institute (2008). *A guide to the project management body of knowledge (PMBOK Guide)*, Project Management Institute, Inc., Newtown Square, Pa.

Robertson, N., and Perera, T. (2002). "Automated data collection for simulation?" *Simulation Practice and Theory*, 9(Copyright 2002, IEE), 349-364.

Ruwanpura Arachchige, J. Y. (2001). "Special purpose simulation for tunnel construction operations." Ph.D., University of Alberta, Edmonton, Alberta, Canada.

Shafaghi, A. (2008). "Equipment failure rate updating-Bayesian estimation." *Journal of Hazardous Materials*, 159(1), 87-91.

Shear, M. (2002). "Springfield Interchange Project Is Defended." Washington Post.

Skoogh, A., Michaloski, J., and Bengtsson, N. "Towards continuously updated simulation models: combining automated raw data collection and automated data processing." *Proc., 2010 Winter Simulation Conference (WSC 2010), 5-8 Dec. 2010*, IEEE, 1678-1689.

Touran, A., and Toshiyuki, A. (1987). "Simulation of tunneling operations." *Journal of Construction Engineering and Management*, 113(4), 554-568.

Xie, H., AbouRizk, S. M., and Fernando, S. "Integrating Realtime Project Progress Input Into A Construction Simulation Model." *Proc., 2011 Winter Simulation Conference, WSC 2011, December 11,2008 - December 14*, Institute of Electrical and Electronics Engineers Inc.

Yang, Z., Moghani, E., AbouRizk, S. M., and Fernando, S. "3D CAD modeling and visualization of the tunnel construction process in a distributed simulation environment." *Proc., 2010 Winter Simulation Conference (WSC 2010), 5-8 Dec. 2010*, IEEE, 3189-3200.

Zeigler, B. P. (1976). *Theory of modeling and simulation*, Wiley Interscience.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation*, Academic press New York.

Zhang, X., Bakis, N., Lukins, T. C., Ibrahim, Y. M., Wu, S., Kagioglou, M., Aouad, G., Kaka, A. P., and Trucco, E. (2009). "Automating progress measurement of construction projects." *Automation in Construction*, 18(3), 294-301.

Zhang, Y., AbouRizk, S. M., Xie, H., and Moghani, E. (2011). "Design and Implementation of Loose-Coupling Visualization Components in a Distributed Construction Simulation Environment with High Level Architecture (HLA)." *Journal of Computing in Civil Engineering*, 1(1), 88.

Zhao, Z. Y., Lv, Q. L., Zuo, J., and Zillante, G. (2010). "Prediction System for Change Management in Construction Project." *Journal of Construction Engineering and Management*, 136(6), 659-669.

Zhou, F., AbouRizk, S. M., and Al-Battaineh, H. (2009). "Optimisation of construction site layout using a hybrid simulation-based system." *Simulation Modelling Practice and Theory*, 17(2), 348-363.

Zhou, F., AbouRizk, S. M., and Fernando, S. "A simulation template for modeling tunnel shaft construction." *Proc., 2008 Winter Simulation Conference, WSC 2008, December 07,2008 - December 10*, Institute of Electrical and Electronics Engineers Inc, 2455-2461.

Zubair, M., Zhang, Z., and Ud-Din, K. S. "Reliability data update method with gamma distribution."