Journal of Simulation (2014) 1–9

www.palgrave-journals.com/jos/

Simphony: a next generation simulation modelling environment for the construction domain

S AbouRizk[1]*, S Hague[1], R Ekyalimpa[1] and S Newstead[2]

[1]*University of Alberta, Edmonton, Canada; and* [2]*SMA Consulting, Edmonton, Canada*

This paper discusses Simphony simulation system in the public domain. Simphony has unique features suitable for solving construction engineering problems. This discussion commences by presenting an overview of simulation, the different simulation methods, and the tools that support these methods. Simphony is then presented as an environment that supports building simulation solutions based on discrete event and continuous simulation. Key features that make Simphony unique are highlighted, with a focus on the ability of the environment to build special purpose simulation tools, an extensible API, an integrated calendar that facilitates building complex models sensitive to 'date and real time' information, data connectivity, and others. A practical special purpose simulator developed using Simphony used in a real life application is presented to demonstrate the use of some of these features.
Journal of Simulation advance online publication, 21 November 2014; doi:10.1057/jos.2014.33

Keywords: Simphony; extensible API; calendars; simulation methods

1. Introduction

Process interaction simulation has been in use for many years in the analysis of complex dynamic systems. The simulation community has a number of well-established methods to apply a simulation-based approach in solving problems: Discrete Event Simulation (DES), Continuous Simulation, System Dynamics (SD), and Agent-Based Modeling.

The use of each of these methods depends on the complexity of the system being analysed and the level of abstraction desired.

DES is an approach in which a system is described using entities, resources, activities, and other modelling constructs. These constructs interact with each other to define the state of the system, and are responsible for its evolution at discrete points in time. In typical DES systems, this change of state is triggered by the flow of entities. Changes in the state of a system typically occur when resources are captured or released, or activities are started or finished. DES is best suited for analysing systems at an operations level. This explains why it has been extensively used in analysing production systems, supply chain, medical facility operations, and construction operations. Various general purpose DES software systems have been developed for a wide range of industries: SimuLink (Mathworks, 1984), AweSim (Pritsker *et al*, 1997), GPSS/H (Crain, 1997), Enterprise Dynamics (INCONTROL Simulation Solutions, 1997), AnyLogic (The AnyLogic Company, 2000), ARENA (Rockwell Automation, 2000), SIMON (Wasshuber, 2001), and Plant Simulation (Siemens PLM Software, 2010), and for construction: Micro-CYCLONE (Halpin, 1973), STROBOSCOPE (Martinez, 1996), and Simphony (Hajjar and AbouRizk, 1999).

Continuous simulation is a branch of simulation in which specific quantities defined by a system (called state variables) are simulated continuously over time. In typical cases, the modeller defines the rates of change of these quantities as a system of differential equations. The simulation engine then implements numeric integration algorithms to generate values for each specific quantity at stipulated points in time.

SD is known for its precision in modelling systems that have numerous components that are dynamic, interrelated, and interact with a feed–feedback behaviour. This method supports a top-bottom approach to that analysis of systems, for example, the evaluation of the impact of different policies or strategies on the behaviour of a system. Simulation systems built to support this method implement numeric integration algorithms (eg, the family of Runge–Kutta equations) in a continuous fashion. It does not involve the flow of tokens, but rather, tracks rates of change in specified quantities with time using integration. Examples of systems that support SD modelling include AnyLogic (The AnyLogic Company, 2000), Vensim (Ventana Systems, 2013), PowerSim (PowerSim Software, 2014), STELLA and iThink (isee Systems, Inc., 2014), Simulink (Mathworks, 1984), and so on.

Agent-based modelling supports a bottom-up approach to the analysis of systems. This approach models a unit or a component within a system as an agent that has intelligent behaviours that are influenced by its peers (other agents) and the environment in which it operates. An agent exhibits different behaviour by transitioning through different states. This behaviour is controlled by an algorithm embedded within the agents. This algorithm is defined using concepts of state diagrams. Communication between agents and the environment is triggered by events

*Correspondence: S AbouRizk, Department of Civil and Environmental Engineering, University of Alberta, 3-015 Markin CNRL Natural Resources Engineering Facility, Edmonton, Alberta, Canada T6G 2W2.

(in the computing science sense). Examples of simulation systems that support this modelling approach include Repast-Simphony, AnyLogic, A3/AAA, ABLE, Agent Builder, MASON, NetLogo, SimAgent, and so on.

2. Simphony simulation system

Simphony is a DES system that was originally developed by Hajjar and AbouRizk (1999). It has evolved over the past 15 years and is currently being extended and maintained by Dr AbouRizk's research team at the Hole School of Construction Engineering at the University of Alberta.

Simphony is built using the Microsoft .NET Framework in a manner that makes it extensible. It consists of three main software modules:

1. Simphony Core Services provides the basic components required to build simulation models. These services include a DES engine, classes for fitting and sampling various probability distributions, statistical services, and so forth. It is possible to build simulation models utilizing only this core portion of Simphony, though it requires expertise in programming with a .NET compliant programming language. This is most often done to incorporate simulation into existing applications such as client server database systems or websites.
2. Simphony Modeling Services is built on top of Simphony Core Services, and provides the components required to develop Simphony templates. Templates are plugins for the Simphony User Interface that provide a set of modelling elements intended to work together. Development of Simphony templates requires expertise in programming with a .NET compliant programming language.
   Most templates developed for Simphony are special purpose templates (SPS), that is, they are designed to model a particular problem domain. These templates provide an effective way to abstract complex processes in a manner that makes it easy for domain experts to make use of simulation without requiring in-depth knowledge of the science behind the method. Examples of special purpose templates that have been developed for Simphony include a tunnelling template, a dewatering template, a programme evaluation and review technique (PERT) template, an earthmoving template, a structural steel fabrication template, and a range estimating template.
   Simphony also provides two templates for generic modelling, that is, they are both capable of modelling processes across a wide range of problem domains. The first is the CYCLONE template that emulates the MicroCYCLONE simulation environment developed by Halpin (1990). The second is the General Purpose template, which is similar to most commercial process interaction modelling systems. The General Purpose template is thus capable of modelling more sophisticated problems. The disadvantage of both of these templates is that the models are more abstract and less representative of the processes they are modelling. This means that users require a deeper understanding of simulation concepts in order to successfully develop models.
3. The Simphony Modeling Environment is the graphical user interface that allows users to develop and run simulation models using the modelling elements provided by one or more templates. The interface permits templates to be added or removed as desired by the user.

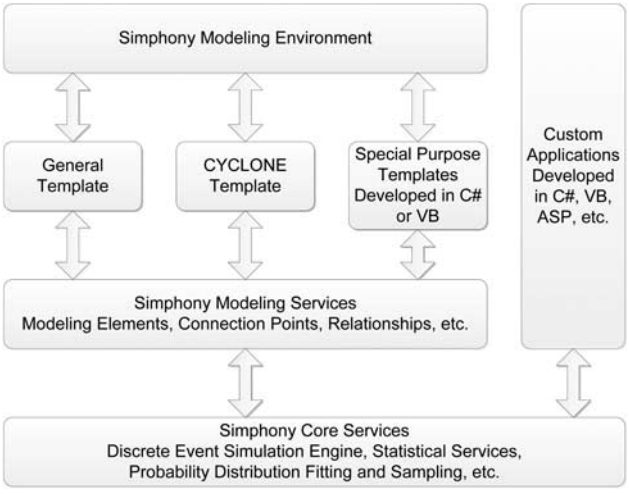

Figure 1 Simphony API structure.

Simphony's structure is illustrated in Figure 1.

Simphony is designed to support Monte Carlo simulation experiments by providing modellers with the ability to define different scenarios for their models, and to specify that each scenario must be run multiple times. Simulation results can then be viewed on a run-by-run basis, or summarized across runs as distributions.

Other features of Simphony include:

- Support of combined discrete event and continuous simulation.
- Support for calendars that define work periods, thus allowing simulation time to be associated with an actual date.
- Supports for connectivity to data storage applications, for example, XML documents or databases.
- Support for integration into larger distributed simulation systems (such as HLA-compliant environments).

The features that make Simphony unique are (1) its ability to create special purpose tools (templates), (2) its ability to modify behaviours of various modelling templates, (3) the integrated calendars for scheduling work, DES, and continuous simulation, (4) a number of templates in the public domain to enable general purpose modelling and special purpose modelling (tunnelling, fabrication, earthmoving, PERT, etc). In this paper we will describe the creation of special purpose simulation tools. The others are well documented (AbouRizk *et al*, 1999; Hajjar and AbouRizk, 1999; AbouRizk and Mohamed, 2000; AbouRizk, 2013).

3. Template development

We now provide an overview of the process of developing a special purpose simulation template for use within the Simphony Modeling Environment. To develop a template, a .NET compliant programming language that can generate a dynamic link library is required. Typically, this will be either C# or Visual Basic hosted within Microsoft Visual Studio. All of the templates mentioned above (including Simphony itself) were developed with Visual Studio. The steps required to develop a template are illustrated in Figure 2.

3.1. Design

The first step of template development is to produce some sort of design document. This will often be a sketch of what a model created using the template might look like together with a document describing the various modelling elements, their purpose, behaviour, and what properties they may have.

3.2. Create entities and modelling elements

Next, a class library project (one that can be compiled into a DLL) is created within the development environment and the classes that will represent the entities and the modelling elements are added to it. In order to be an entity, a class must ultimately derive (in the object oriented programming sense) from the Entity class (which is found in Simphony Core Services). Similarly, in order to be a modelling element, a class must derive from the ElementBase class (which is found in Simphony Modeling Services). It is not uncommon for entities to derive directly from the Entity class; however, it is rare for modelling elements to derive directly from the ElementBase class. More often, modelling elements derive from intermediate classes provided by Simphony Modeling Services, which in turn derive from the ElementBase class. An example would be the FlowElement class, which provides a single input point for entities to flow in and a single output point for entities to flow out.

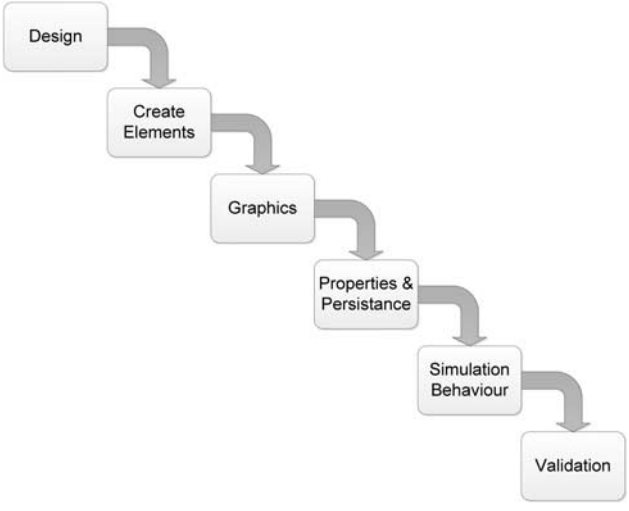

Figure 2 Template development process.

Once these classes have been added to the development project, graphical icons need to be developed to represent both the template and the modelling elements on the Template Palette inside the Simphony Modeling Environment. These icons are added to the development project and then associated with the corresponding class.

3.3. Graphics for modelling elements

Once the required modelling elements have been defined, the next step is to write the code that will render the modelling elements on the Modeling Surface inside the Simphony Modeling Environment. There are a number of ways that modelling elements may be rendered. Two of the most common are:

1. Overriding a modelling element's Paint method and making calls to GDI+ (Microsoft's 2D graphics library) to draw the modelling element piece-by-piece.
2. Using either a bitmap image (BMP, PNG, JPG, etc) or a vector image (WMF, EMF, etc) to represent the modelling element.

3.4. Properties and persistence

After a modelling element's graphics have been defined, the properties that appear in the Property Grid of the Simphony Modeling Environment must be added. This is done by adding the appropriate properties to the class representing the modelling element. Writing the code for properties is quite simple if the underlying data type is simple (eg, integer, double, or string); however, if the property's data type is more complex, say, for example, a collection, then more work is required: classes representing the items stored in the collection will need to be developed and user interfaces may also be required to permit the end user to edit those items.

In addition to the properties themselves, the template developer must override the WriteXml and ReadXml methods of each modelling element and add code that defines how the values of the properties will be persisted inside a Simphony document. Simphony stores models in XML formatted files that are given the SIM file extension, so a good understanding of XML (Extensible Markup Language) is required.

3.5. Simulation behaviour

The penultimate step of template development is to add the modelling element's simulation behaviour. The template developer must write code to define what happens when an entity arrives at a modelling element's input port. If the entity requires a shared resource to perform the activity the modelling element represents, then that resource needs to be defined (along with an associated queue) and the entity must make a request for that

resource. Decisions may need to be made based on what other entities are doing or what resources are available. Events may need to be scheduled or processed. Entities may need to be routed out of a modelling element's output port. All of these actions are handled using services provided by the Simphony Modeling Services or Simphony Core Services.

3.6. Validation

Finally, once the template has been developed, it must be validated to ensure it is working as intended. Many of the validation techniques described by Sargent (2003) are applicable to template validation. Two of the most common are:

1. *Comparison to other models*: It is often the case that valid general purpose models (developed using the General template or the CYCLONE template) exist in the intended problem domain of the template. In these cases, the new template can be used to model the same problem and the results compared.
2. *Predictive (or historical) validation*: The new template is used to model an existing (or historical) system, and the behaviour of the template is compared with that of the system.

4. Template development case study

We now present Simphony's Tunneling template as a case study in template development.

Obviously, before beginning any development work, template developers must familiarize themselves with the problem domain. For tunnelling operations using a tunnel boring machine (TBM), the first step is to excavate a shaft with a work area at its base (called an undercut) that is used to install the TBM at the depth of the tunnel. Once installed, the TBM begins excavation of the tunnel. It does this in a cyclic process that consists of three steps: excavation of a 1 m segment, installing concrete liners to support the excavated segment, and resetting in preparation for excavating the next segment. As the TBM progresses, tracks are installed in the tunnel and two trains are used to haul the spoil back to the undercut and to bring concrete liner segments out to the TBM. The spoil is brought to the surface by a crane that is also responsible for lowering the concrete liner segments. Trucks then haul the spoil away from the site. While the TBM is excavating, a second shaft will be dug at the end point of the tunnel. This second shaft will be used to remove the TBM when the tunnel excavation is completed.

4.1. Design

The first step in development is to draw a sketch of what the tunnelling process might look like when hosted in the Simphony Modeling Environment, as shown in Figure 3. As can be seen from the sketch, the template consists of four primary modelling elements:

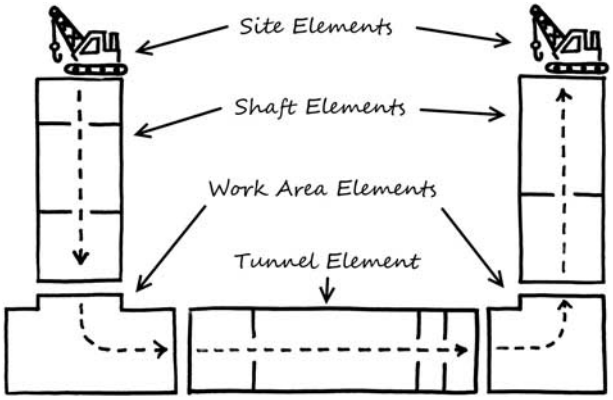

Figure 3 Sketch of the tunnelling template.

1. The Site element is responsible for modelling the work site on the surface. It models the crane, the spoil pile, the trucks hauling the spoil away, and the inventory of concrete liner segments. At the end of simulation it should be able to report on the utilization of the crane and the size of the spoil pile over time. In a typical tunnelling project, there will be two site elements: one located at the working shaft and the other located at the removal shaft.
2. The Shaft element is responsible for modelling construction of shafts. It supports various construction methods (continuous drilling, rib-by-rib drilling, hand excavation, etc) and includes information about geotechnical conditions so that excavation rates and swell factors can be modelled accurately. The Shaft element is able to model both working shafts and removal shafts.
3. The Work Area element is responsible for modelling both the construction of the undercut and the unloading/loading of trains during tunnel construction. The same element is used to model the working area at the base of the removal shaft, though in this case, only construction of the work area need be modelled.
4. The Tunnel element is responsible for the activities associated with the TBM. Thus, it models the excavation process, the unloading of liner segments from trains, and the lining and resetting processes. Like the Shaft element, the Tunnel element includes information about geotechnical conditions so that excavation rates and swell factors can be modelled accurately.

In addition to the primary modelling elements, there are a number of additional requirements for the template:

- It should make use of Simphony's calendar services so that it can produce a schedule that can be exported to Microsoft Project.
- The various modelling elements should contain cost information (labour, equipment, materials, etc) so that a cost estimate can be produced.

- The template should support Monte Carlo simulation, so that durations and production rates can be expressed as probability distributions.
- It should have the ability to define breakdown intervals and repair times for the different types of equipment in use: the crane, the trains, and especially the TBM.

4.2. Create entities and modelling elements

Once the template design has been completed, coding can begin. The template was developed in C# using Microsoft Visual Studio. A Windows Class Library (DLL) project was created, and classes added to represent the four primary elements. Each of these classes derives ultimately from the ElementBase abstract class defined in Simphony Modeling Services. It was decided at this time that the Shaft and Work Area elements would be composite elements; that is, they would be able to contain other (more general purpose) modelling elements, thus creating a hierarchical structure. This decision was made so that users would be able to heavily customize the construction processes of both the shaft and work areas. Icons (16 × 16 pixels) were then created to represent each of the modelling elements in the Template Palette.

4.3. Graphics for modelling elements

It was decided that a JPEG image of a typical crane would be used to represent the Site element. The remaining elements were drawn using calls to GDI+ so that the various soil layers can be shown graphically. In addition, arrows will be drawn (as depicted in Figure 3) indicating the direction of travel of the TBM. It was also decided that the modelling elements will not be drawn to scale with each other as in typical tunnelling projects the tunnel is considerably longer than the shaft is deep. Implementing the graphics for the Site element is relatively easy, as the only work required is finding a representative image. The remaining elements need to be coded in C#, and in the case of elements incorporating soil layers, some of the coding will have to be done after the corresponding properties (discussed in the next section) have been implemented.

4.4. Properties and persistence

Each of the primary modelling elements requires a large number of properties. For space considerations, we will only discuss those of the Shaft element.

The Shaft element has four simple input properties: its shape (circular *versus* rectangular) together with its diameter (for circular shafts) or length/width (for rectangular shafts). It also contains a collection property that defines one or more soil layers. Each soil layer is a complex type (ie another class) that defines a depth property, a soil type property, and a swell factor property. A UML class diagram illustrating this structure is shown in Figure 4. All of these properties together with the soil layer class itself needed to be created using C#.

In addition to creating the properties, both the Shaft element and the soil layer class needed to be made XML serializable; that is, the two classes needed to know how to read themselves from and write themselves to an XML document. A modelling element is made XML serializable by overriding the ReadXml and WriteXml methods it inherits from the ElementBase class. The soil layer class is made XML serializable by implementing the IXmlSerializable interface provided by the .NET framework.

4.5. Simulation behaviour

The final step of coding the template is adding the simulation behaviour. Again, this requires a fair amount of effort for each element, so for space considerations we will only discuss the Tunnel element.

The Tunnel element defines two resources: the track and the TBM, each of which has only a single server. It also defines waiting files (queues) associated with each of these resources. During the tunnel construction process, entities representing the trains are routed into the Tunnel element from the Work Area element that represents the working undercut. When an entity first arrives, it requests the track resource. If another train is currently in the tunnel, it will be forced to wait until the other train leaves the tunnel and frees the track resource; otherwise it will be granted the track resource immediately.

Upon obtaining the TBM resource, the train entity can begin the excavation activity and the unloading of liners activity. Both of these are modelled as events scheduled on the event queue. The duration of the excavation activity is stochastic and also depends on the current soil conditions; the duration of the unloading activity is constant. Once both activities are completed, the TBM resource is freed.

At this point, the train entity begins its journey back to the working undercut. As before, this is modelled by scheduling an

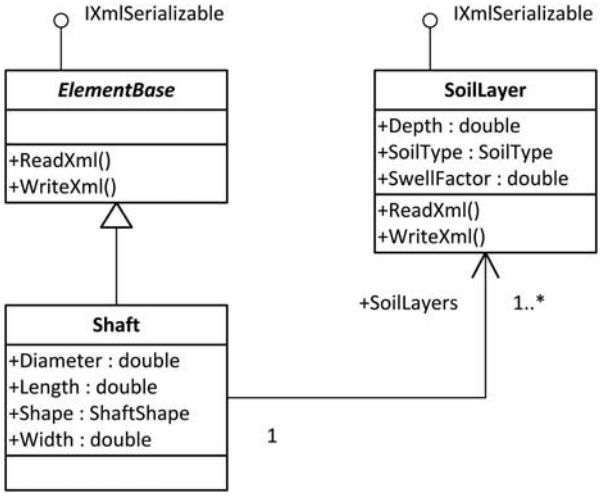

Figure 4 UML class diagram for the shaft element.

event at the time the train will arrive at the undercut. In parallel with the train's return journey, a new entity is created and requests the TBM resource. Once granted the TBM, this new entity will model the lining and resetting processes of the TBM. When these two processes are complete, the entity frees the TBM resource and is destroyed.

A UML activity diagram illustrating the simulation behaviour of the Tunnel element is shown in Figure 5. All of these processes needed to be written in C# and frequent use was made of Simphony Core Services.

5. Simphony application: implementation on a practical problem

To demonstrate the practicality of Simphony's Tunneling template, a real-life tunnel project, to which Simphony was applied for analysis, is detailed.

5.1. Project description

The project is a 708.0 m tunnel (excluding the tail tunnel and undercuts at the working shaft and retrieval shaft), to be constructed using a 2.6-m-diameter M100 TBM. The geological soil profile along this tunnel is summarized in Figure 6.

The working shaft is composed of three soil layers: topsoil, clay fill, and clay. It is approximately 8.0 m deep, 4.5 m in breadth, and has been assumed to be completed as a rib-by-rib, drill-assisted excavation. The removal shaft has a 12.8 m total depth, is 4.5 m in breadth, and has the following soil layers: asphalt, gravel fill, clay fill, clay, clay till, and sandstone. The first 6.0 m of the shaft have been assumed to be excavated continuously by the drill rig; the remaining 6.8 m have been assumed to be excavated as rib-by-rib, drill-assisted excavation.

The working undercut is situated in a clay soil layer. The working undercut was assumed to be excavated in a two-way hand tunnel excavation method, such that both directions of hand tunnelling are conducted at the same time. The tail tunnel portion of the undercut is 13.0 m in length, while the TBM undercut portion is 12.0 m in length, and both are situated in a clay layer.

5.2. Simulated scenario

A simulation model was developed for the described project using the tunnel template. The model layout is summarized in Figure 7.

Before simulating the model, a number of inputs were entered into the model. These included tunnel geometric information, geologic information, equipment information (failure, production rates, and unit costs), and other resources. Equipment and resource information were based on historic data collected on completed projects. The work shift to be simulated was defined within the calendars.

A double 8-h working shift was defined and simulated for 10 runs. The total cost estimate for the project estimated at the

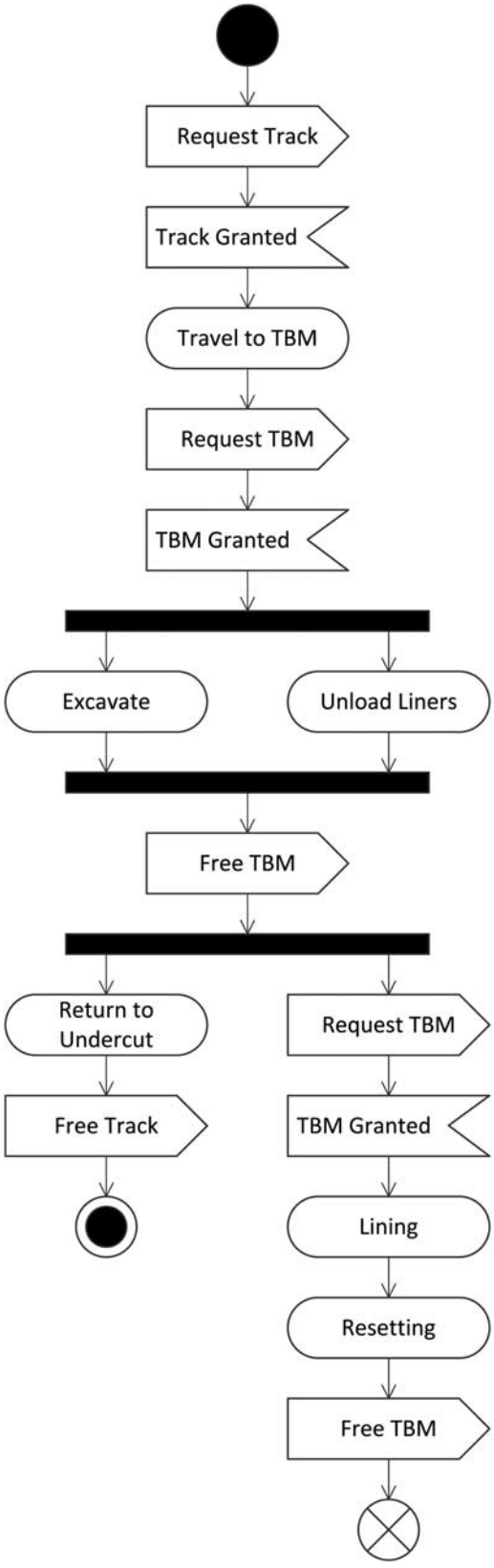

Figure 5 UML activity diagram for the tunnel element.

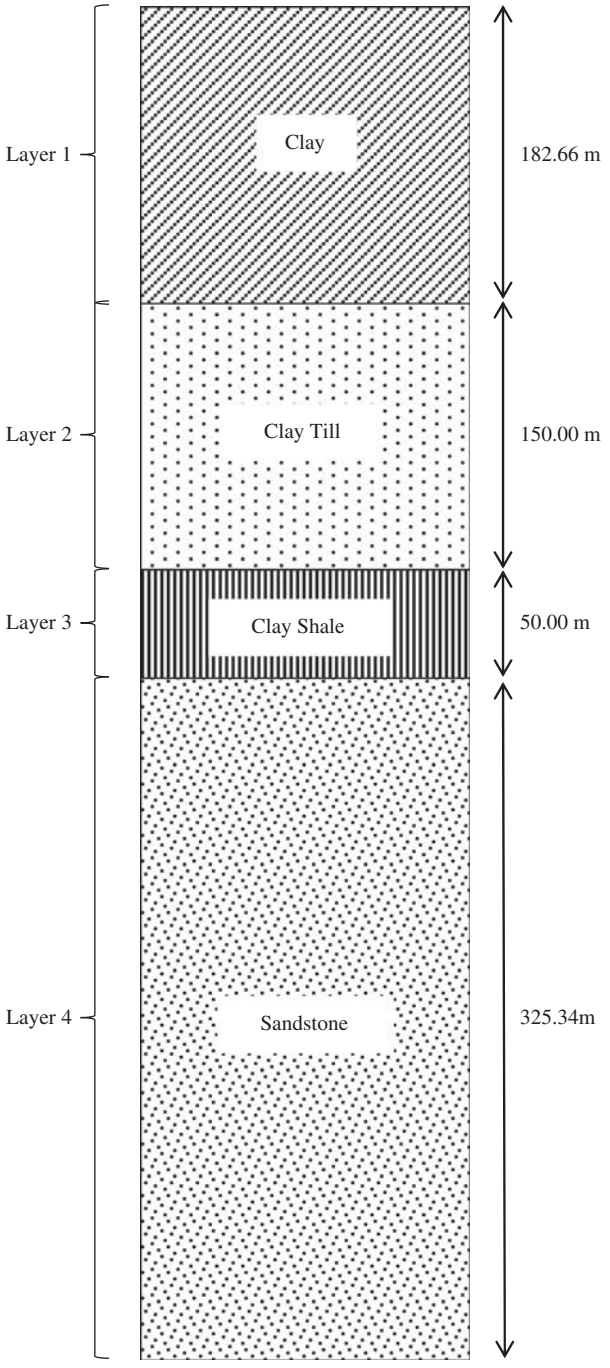

Figure 6 Soil layers along the tunnel from working shaft to retrieval shaft (top to bottom in the figure).

90th percentile was US$11 775 385.36, approximated as follows:

$$X_q = \overline{X} + z_q S = 11\,650\,026 + 1.28 \times 97\,937$$
$$= \$\,11\,775\,385.36$$

The simulation also reported that with this double 10-h working shift, the entire project would be completed in 7298.02 h (combined regular time and overtime) at the 90th percentile. This

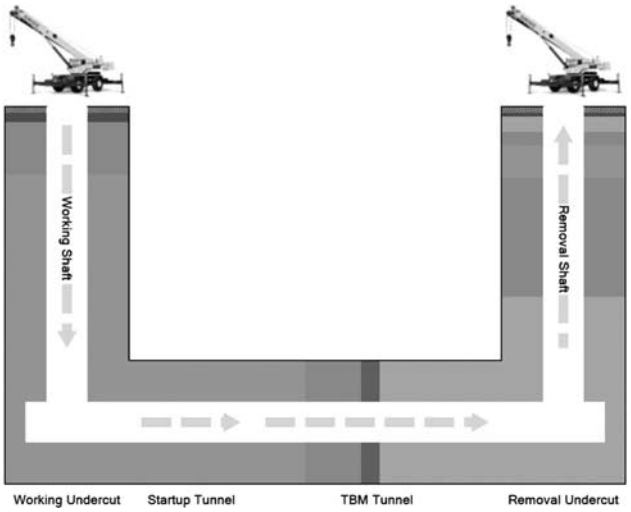

Figure 7 Model layout developed using the tunnel template within Simphony.

was estimated using the equation below.

$$X_q = \overline{X} + z_q S = 7187.22 + 1.28 \times 86.56 = 7298.02\,h$$

5.3. Use of the case study to verify and validate the template

The tunnelling template was validated in two ways. First, an assessment was done of concept models used to abstract and formalize the problem. The second involved a comparison of template results with data obtained from site on a real tunnelling project. In between these two exercises, extensive verification work was also done.

As part of the template development process, a number of concept models were created similar to that shown in Figure 5. These help in the abstraction and formalism of the real life tunnelling operation. In order to ensure that these concept models accurately match real life processes, they were discussed, reviewed, and updated through consultation with tunnelling experts. After a series of reviews and updates, the experts confirmed that the concept models accurately represented the tunnel construction process. This, according to Boehm (1981), meant that the concept models are valid, marking an important milestone in the development of a valid template.

Thereafter, the template was developed. This development work was tightly coupled with verification activities to ensure that the concept models were accurately translated to computer simulation constructs (modelling elements) by programmers. Verification included: (1) generating trace logs of simulated processes, (2) performing unit tests using results from hand simulated computations in Microsoft Excel, and (3) a comparison of results generated from general purpose template tunnel models built off of the same logic summarized in the concept models. The results from the unit tests and general purpose template models were found to match those from the template. Also, the details generated from the simulation within the trace logs were

found to follow the logical sequence detailed in the concept models. This proved that the concept models were accurately translated into a template, hence, verifying the tunnelling template.

As a way of confirming the validity of the concept models and the tunnelling template developed from these models, validation was done using data collected on a real tunnelling project. Details of this project are presented in Section 5.1. Fortunately, the construction of the project was underway at the time of writing this paper. The working shaft, tail tunnel, and undercut (both at the working shaft) have all been fully excavated. The TBM had also been fully installed and was ready to go. The installation of concrete liners and associated backfilling remains to be done at the closing stages of the entire project. Data obtained from site on these three components were used in the validation of modelling elements that mimic their construction.

Costs were used to test the validity of the template. Simulated and actual costs for these components are summarized in Table 1.

These number translate into an average error of 5.06%. There are not strict guidelines on error thresholds that distinguish valid from invalid models. However, the general rule is that the smaller the error, the better. A value of 5.06% is not significantly large, and consequently, it can be concluded that the tunnelling template is valid. The same applies to the concept models that were used to develop this template.

5.4. Discussion

Two case studies were presented in this paper: one summarizes the development of the tunnel template and the other illustrates the use of this template. The first case study showcased how the special purpose template development steps are systematically applied to develop a specific template within the Simphony simulation environment. The tunnel template is used as an example for this. The case study demonstrated (1) the ease with which templates can be developed in Simphony by strictly following these systematic steps, and (2) the accuracies that can be obtained as a result.

The second case study demonstrated how to use the special purpose template to solve a real world tunnel construction problem. Before the construction of any tunnel, the practitioners tasked with setting up the construction operation have ample information at their disposal, for example, the product model of the tunnel, geotechnical data, other ground surface conditions related to the site, options of excavation methods, and options for different shift configurations.

Utilizing this information in an effective fashion, without design aides, to search for an appropriate construction operation setup that will deliver the desired targets (budget and schedule), is extremely challenging, even for very experienced practitioners. This is because there is a lot of information to synthesize in order to arrive at a decision, and yet, most aspects of the problem are also highly stochastic, dynamic, and interrelated in a non-simplistic fashion. In addition, replicating successes from past projects is not feasible. For example, the setup for the material handling process, that is, the number of carts per train (and whether to use an undercut or not), the number of trains (whether to construct a switch or not), depends on a number of variables, such as the total length of the tunnel, the point along the tunnel at which excavation is being done, the geological formations along the route of the tunnel, and the capacity (penetration rate) of the TBM. The ideal situation is to have the material handling cycle(s) balanced with the excavation and liner erection cycles in order to minimize idling of resources.

This second case study demonstrates how the tunnelling template can be used to simplify the analysis of such a complex operation. The case study also demonstrates the fact that, once a special purpose simulation template has been developed for a particular domain, setting up models for unique projects/problems in that domain and experimenting with these models does not require a lot of time and effort. This convenience in using special purpose templates is attributed to the philosophy behind their development and usage, that is, embed most of the process logic within the coded simulation behaviour of the modelling elements so that the overall model layout development using these templates is not cumbersome. Last, the case study demonstrated how, in a practical setting, relevant data can be filtered out of the bulk of information available to practitioners, then used as input for special purpose template models in simulation-based analysis.

Table 1 Actual and simulated costs for the shaft, undercut and TBM installation

Simulated (Experiment #)	*Shaft ($)*	*Undercut ($)*	*TBM Installation ($)*
1	419 605.42	865 400	223 499.09
2	422 437.46	770 000	222 405.53
3	421 889.59	830 000	223 798.81
4	416 459.96	765 000	220 713.83
5	419 504.89	930 000	234 638.12
Actuals	355 186.76	703 053.80	311 421.12

6. Conclusions

This paper discussed Simphony simulation system in the public domain. In particular, Simphony's unique features, which make it ideal for solving construction engineering problems, are discussed. These key features include the ability of the environment to build special purpose simulation tools, its extensible API, its integrated calendar that facilitates building complex models sensitive to 'date and real time' information, data connectivity, and others. A practical problem (a real-life tunneling project) modeled in a special purpose simulator, developed using Simphony, is described to showcase capabilities and features that exist within Simphony.

References

AbouRizk S (2013). Innovations in tunnelling construction management: Applications of simulation. In *Proceedings of the 3rd International Conference on Computational Methods in Tunnelling and Subsurface Engineering*, Aedificatio Publishers: Freiburg, Baden-Württemberg, Germany.

AbouRizk S and Mohamed Y (2000). Simphony—An integrated environment for construction simulation. In *Proceedings of the Winter Simulation Conference*, IEEE: New Brunswick, NJ, pp 1907–1914.

AbouRizk SM, Ruwanpura JY, Er KC and Fernando S (1999). Special purpose simulation template for utility tunnel construction. In *Proceedings of Winter Simulation Conference* (2). IEEE: New Brunswick, NJ, pp 948–955.

Boehm BW (1981). *Software Engineering Economics*. Prentice Hall, Inc.: Upper Saddle River, New Jersey.

Crain RC (1997). Simulation using GPSS/H. In *Proceedings of Winter Simulation Conference*, IEEE: New Brunswick, NJ, pp 567–573.

Hajjar D and AbouRizk S (1999). Simphony: An environment for building special purpose construction simulation tools. In *Proceedings of Winter Simulation Conference* 2. IEEE: New Brunswick, NJ, pp 998–1006.

Halpin DW (1973). *An investigation of the use of simulation networks for modeling construction operations.* PhD thesis, University of Illinois, Urbana-Champaign, IL.

Halpin DW (1990). *MicroCYCLONE System Manual*. Division of Construction Engineering and Management, Purdue University: West Lafayette, IN.

INCONTROL Simulation Solutions (1997). *Enterprise Dynamics* (Version 1.0) [computer program]. INCONTROL Simulation Solutions: Utrecht, The Netherlands.

isee Systems, Inc (2014). *STELLA* [computer program] isee Systems: Lebanon, NH.

isee Systems, Inc (2014). *iThink* [computer program] isee Systems: Lebanon, NH.

Martinez JC (1996). *STROBOSCOPE—State and resource based simulation of construction processes*. PhD thesis, University of Michigan.

Mathworks (1984). *SimuLink* (Version 1.0) [computer program] Mathworks: Natick, MA.

PowerSim Software AS (2014). *PowerSim Studio 9* [computer program]. PowerSim Software AS: Bergen, Norway.

Pritsker AAB, O'reilly JJ and Laval DK (1997). *Simulation with Visual SLAM and AweSim*. John Wiley & Sons, Inc.: New York.

Rockwell Automation (2000). *ARENA* [computer program]. Rockwell Automation: Wexford, PA.

Sargent RG (2003). Verification and validation of simulation models. In *Proceedings of Winter Simulation Conference*. IEEE: New Brunswick, NJ, pp 37–48.

Siemens PLM Software (2010). *Plant Simulation* (Version 1.0) [computer program]. Siemens PLM Software: Plano, TX.

The AnyLogic Company (2000). *AnyLogic* [computer program]. The AnyLogic Company: St. Petersburg, Russia.

Ventana Systems (2013). *Vensim* (Version 6.2) [computer program]. Ventana Systems: Harvard, MA.

Wasshuber C (2001). *SIMON* (Version 1.0) [computer program]. Cambridge, MA.

Received 25 February 2014;
accepted 23 October 2014 after one revision