Object-Oriented Systems Engineering for Infrastructure Projects

Ian Brace

Shoal Group Pty Ltd

ian.brace@shoalgroup.com

ABSTRACT

Object-oriented systems engineering is a model-based approach that is typically associated with SysML and is recommended for software intensive projects. However, the underlying object-oriented concepts, such as inheritance, encapsulation and component reuse, all have great utility on the class of projects that deliver similar functionality across multiple physical sites. This is especially true of infrastructure projects such as railway networks, surveillance networks and other transport networks.

Infrastructure projects are not usually software-intensive in the way that military or aerospace projects might be. Therefore, systems engineers in these domains are less likely to be familiar with SysML or perhaps even object-oriented concepts. When introducing these ideas, it would be advantageous to minimise the barriers to adoption by reducing the scope of the learning curve. This paper describes how object-oriented techniques can deliver MBSE advantages in the infrastructure domain without demanding the adoption of SysML. The approach is based on the principle that the barriers to entry into MBSE should be as low as possible.

The paper starts by reviewing the key advantages of object-oriented techniques as applied to infrastructure projects, such as abstraction, inheritance, encapsulation and model reuse. It then compares the object-oriented approach to traditional structured analysis and affirms that both approaches are valuable, depending on the type of project being undertaken. A worked example of a network infrastructure projects that benefit from an object-oriented approach is discussed. The paper illustrates the use of abstraction to establish a logical architecture and the use of inheritance to create a physical architecture that represents the nodes of the deployed system. Encapsulation is illustrated through the logical components and alternate architectures are presented to illustrate the benefits of component reuse across the model.

INTRODUCTION

Object-oriented systems engineering emerged in the late 1990s through a desire to extend the advances made in object-oriented software development to general system design. Although object-oriented concepts were sometimes applied to socio-technical systems (Bharathan, Poe and Bahill), the main motivation was to develop a common approach on software-intensive projects between software engineers and systems engineers. For this reason, object-oriented systems engineering is often associated with the OMG[1] System Modelling Language (SysML) which was first released in 2007. SysML is useful when developing complex projects because it helps to specify system components which can then be designed using other domain-specific languages such as UML[2] for software components and VHDL[3] for hardware design (Friedenthal, Moore and Steiner). SysML is a natural companion to Model-Based Systems Engineering (MBSE) but it is not analogous to it.

The Object-Oriented Systems Engineering Methodology (OOSEM) was developed through a continuing INCOSE working group and the method is referenced in Systems Engineering Handbook (INCOSE TP-2003-002-04). OOSEM was specifically developed to use SysML (Friedenthal, Moore and Steiner).

[1] Object Management Group

[2] Unified Modelling Language

[3] Very High Speed Integrated Circuit Hardware Description Language

Despite its origins in software and complex systems within the defence and aerospace domains, object-oriented systems engineering has strong utility in other industries. Infrastructure projects that combine complex systems with construction can benefit from object-oriented design. Such projects regularly occur across many industries including public transport, construction and even defence.

However, as a design methodology systems engineering has had less exposure within these other industries, much less object-oriented systems engineering. It is impractical to simultaneously ask engineers to adopt systems engineering; learn a new MBSE tool; apply object-oriented concepts; and learn a new modelling language. The barriers to introducing a SysML-based object-oriented methodology into new industries are very high, particularly when software design is not a strong discipline within that industry.

This paper proposes an MBSE approach that introduces some of the key benefits of object-oriented design to infrastructure projects without demanding a SysML solution. It retains the benefits of traditional systems engineering structured analysis while introducing abstraction, inheritance and component reuse to simplify infrastructure solution design. The approach seeks to exploit the advantages of object-oriented design while minimising the entry barriers to engineers.

OBJECT ORIENTED SYSTEMS ENGINEERING

Concepts

Object-oriented techniques are well established in the software domain. It is a view of programming in which data and behaviour are strongly linked. Data and behaviour are conceived of as abstract classes which are instantiated as objects (Pohl). An object can receive certain types of data and can process them in particular ways, leaving the object in a particular state. The details on how the processing is performed is hidden from view within the object; the user only sees data going in and the processed data coming out.

The extension to systems engineering is attractive. A system comprises a set of components, or objects, and during a design cycle we can imagine them as *black-box* components. In this context *black-box* means we have defined *what* functions they will each perform, but not *how* they will execute their functions. Each component accepts particular inputs, based on an interface functional specification, and the component functions produce the required outputs. Component signals and functions are strongly linked. Once the system behaviour, as defined by component interaction, is understood we can decompose each component into sub-components and repeat the design cycle.

There are four key features of object-oriented methods that have utility in systems engineering: Abstraction; Encapsulation; Inheritance; and Re-use.

Abstraction

This feature allows us to abstract a component from its physical implementation and instead focus on its logical representation. Logical components often describe functionality and component attributes that are common across the system. By removing specific physical constraints, we can better understand what the component must do and how it interfaces with other logical components.

Encapsulation

Objects have a well-defined boundary with specific interfaces to the remainder of the system. The functions of the object are ‘encapsulated’ within the object and are not visible to external components. Instead, external components interact using the defined interfaces that the component makes available. Encapsulation improves modularity and reduces the inter-dependency between components. This means a component technology can be replaced as long as the replacement maintains the same functional outputs and the same set of interfaces.

Inheritance

Inheritance is a powerful attribute that allows one component to inherit all of the functionality and interfaces of another component. In object-oriented systems engineering inheritance is typically used when creating a representation of a physical component. The physical component can then ‘inherit’ a logical component that has already been designed. However, inheritance is not like cloning. Once the

physical component has inherited an abstract parent component, the engineer can also add additional functions, requirements or interfaces specifically for that instance of the physical component. A child component can also inherit from more than one parent, which provides a flexible way to build up complex designs.

Object Re-use

Following on from inheritance, a parent can be inherited by multiple children if they all perform a largely common set of functions but then need some specific requirements that are not all the same. This provides efficiencies, since the engineer only needs to design the parent once, but can reuse the component in multiple places.

STRUCTURED ANALYSIS VS OBJECT-ORIENTED ANALYSIS

Traditional systems engineering has used what is commonly called *structured analysis* to analyse user needs and develop a system model. Both structured analysis and object-oriented analysis aim to to define a modular system design, but they use quite different principles to achieve it.

Structured analysis is characterised by its orientation on the functions to be performed by the system (Van Assche). Functional analysis is central within the process to deliver a comprehensive system behaviour model and a corresponding data flow. Functions are characterised as data transformations, operating on input sets to produce particular outputs (Wymore). In structured analysis, once the behaviour of the system is understood then the functions are packaged into components and the design moves towards the physical model. This approach is neatly illustrated as a process flow by the ubiquitous systems engineering V-diagram. Of course, in practice it is never a simple flow; there are iterations across each boundary as we move down the design prcoess.

In contrast to structured analysis, object-oriented analysis is characterised by its orientation on the logical components that perform the system functions. It turns the order of design around; rather than delaying the definition of the components until after the system behaviour has been defined, object-oriented analysis defines the components early and then, during the first design cycle, allocates the necessary functions that the component must perform within a successful system. This approach seeks to encapsulate the state and behaviour of the components (Van Assche). During the next design cycle each component object is decomposed into sub-components and allocated sub-functions. Together, these sub-components must deliver the performance of the top-level component. A representation of these two approaches is presented in Figure 1.

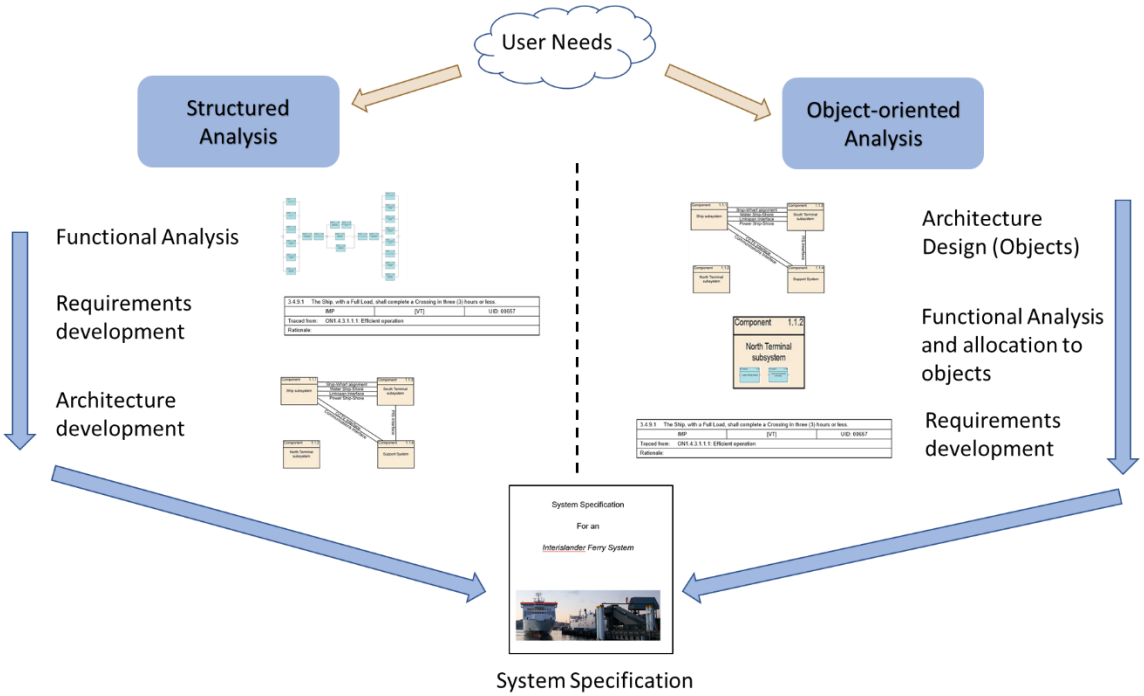

Figure 1 - SA vs OOSE

Arguments can be mounted to support both approaches. By stressing functionality and *solution independence* structured analysis enforces a discipline that reduces the chance of producing a design that replicates an existing architecture. Object-oriented analysis argues that specifying the model independently from the functions results in a more intuitive model which will be more understandable to the user (Van Assche). A pragmatic approach is to consider the project at hand and select a design methodology that supports the relevant engineering context.

OBJECT-ORIENTED TECHNIQUES IN INFRASTRUCTURE PROJECTS

Infastructure projects have not historically been characterised as software intensive. Times are certianly changing, but software and hardware integration has not typically been a key aspect of an infrastructure project. This is not to suggest that software does not play a part in such projects, but it has not usually been a defining characteristic. The engineering disciplines across infrastructure projects are also not the typical electronic, aerospace, software or mechanical disciplines found in the defence and aerospace domains.

Whilst sysems engineering and MBSE is gaining traction within infrastructure industries, exposure to object-oriented concepts is less widespread. Nevertheless, abstraction, inheritance, encapsulation and re-use are useful concepts in an infrastrucutre context.

Network infrastructure projects often build nodes in multiple locations that together form a network capability. There can often be commonality of *function* across nodes, but inevitably the physical works required in each location will be unique to that particular node. This aligns with the object-oriented concept of an abstract parent class that encapsulates the common functionality which is then inherited by each physical instantiation of a particular node. The core functions remain common across all nodes, but each child node is then supplemented with additional functions and attributes specific to that location.

Using an object-oriented methodology also supports exploration of multiple solution architectures. Network infrastructure functionality can be modelled as a single logical layer, while multiple physical architecture options can be modelled as separate physical layers. Using object-oriented techniques, the various logical elements can be inherited by different physical infrastructure elements within each propsoed solution. Trade-off studies can then determine the optimal physical architecture to deliver the necessary function and performance across the network.

OBJECT ORIENTED MBSE WITHOUT SYSML

MBSE enhances Systems Engineering

The benefits of MBSE have been well documented in the literature (Honour) (Murray). Some of the key benefits are traceability, sytematic structure, multiple views of data and automatic reporting.

a. **Traceability**. Systems engineering models provide explicit traceability from strategic and operational needs down to system requirements and their verification activities. Across the design process, MBSE tools largely automate the review of traceability which provides assurance that the design has addressed the need.

b. **Systematic structure**. MBSE models have inherent data structure through the databse schema, which enforces design discipline. MBSE tools usually highlight where relationships have been omitted, which reminds engineers to re-examine the surrounding issues and resolve the gaps in the model.

c. **Multiple data views**. MBSE tools make it easy to view a systems engineering model from multiple viewpoints. The defined relationships between the model elements allow the data to be manipulated to present relevant views of the system to different stakehodlers.

d. **Collaboration**. MBSE tools make data access easier and allow teams of engineers to collaborate around the same relevant data.

MBSE Model Design

All MBSE models are built on an entity-relationship database using an agreed schema to define the entities, their attributes and the relationships between the different entities. However, the breadth of the

out-of-the-box schema varies between tool vendors, largely based on the design philosophy under pinning the tool.

Tools that support SysML as the design language have a basic set of entities with relationships that comply with the SysML specification. The entities have generic names, such as *Block* and *Activity*. The design philosophy expects engineers to take these generic entities and categorise then, or label them, in a way that is suitable for the specific design project being undertaken. For example, an *Activity* may represent a system function, or an *Activity* could be categorised as an operatioanl activity performed by an end-user.

In contrast, methodologies implemented by other tools (Estefan) provide a very rich standard schema that predefines all of the common entities typically found in a complex system design and underpins the MBSE approach. Ready-made entities can include *Operatioanl Activities* and *Operational Needs, Functions, Components, Interfaces, Verification Activities, Risks* and a host of other common artefacts, and the relationships between them, that are used across the design cycle. Of course, all of these entities could also be defined using SysML, but they are not generally provided out-of-the-box.

Within industries that have less exposure to systems engineering it is easier to introduce tools that have a detailed schema. Whilst SysML can certainly be made to mirror such a schema, asking inexperienced systems engineers to build the schema introduces yet another barrier to adoption of the systems engineering methodology.

In this paper we will develop an example of a rail network model, using a tool called *Capability Architect*[4] which can support object-oriented systems engineering but does not demand the use of SysML.

MBSE Model Structure

Key schema entities

The *Capability Architect* schema is comprehensive, and includes the following traceability relationships:

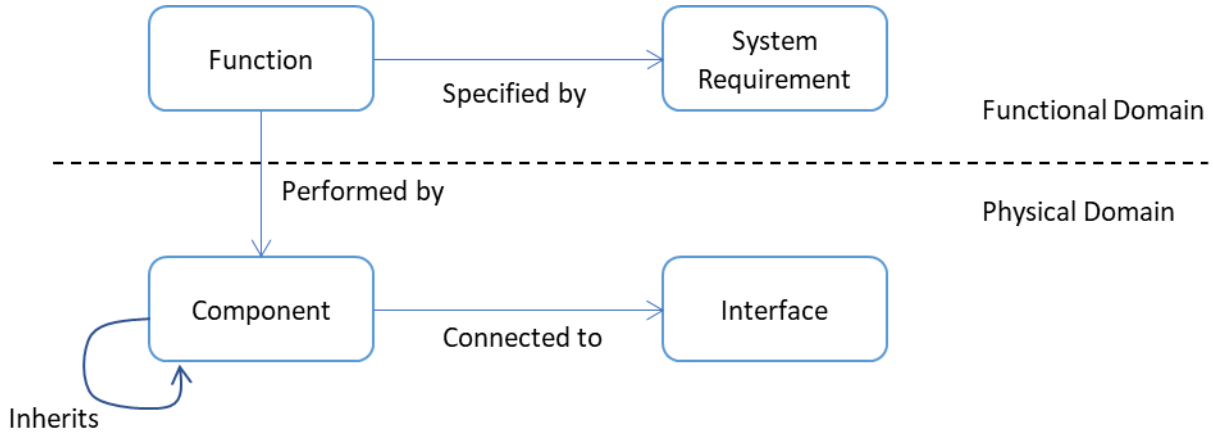

Figure 2 - Capability Architect schema subset

Object-oriented concepts are most effectively applied in the 'Physical Domain', using the Component class. We can separate the Component class into *Logical Components* and *Physical Components*.

WORKED EXAMPLE: DESIGNING A RAIL NETWORK

A rail network can be characterised as a set of network elements, rolling stock and platform elements. Although each platform will have particular build requirements, based on its location and geography, the core functions of a platform are to deliver a common set of services to passengers and train drivers.

An object-oriented approach to modelling the rail network would define a *platform* as an abstract class that encapsulates the common functions. This abstract component would be part of the network *logical*

[4] https://systemdesigngroup.com.au/capability-architect/

architecture that also defines the interfaces between the platform and the other logical elements, such as a *train* and the *signal system*.

The *physical architecture* would contain a set of components that denote the deployed rail network, such *Prahan Station, Frankston Station*, etc. Each physical component inherits a *platform*, which immediately invests it with the common set of functions and the common interface requirements. Additional unique requirements can then be added to each specific *station* component.

By modelling this concept within an MBSE tool, engineers can then easily produce multiple design artefacts. A *System Description Document* that focusses on the operation of the system can report against the elements in the logical architecture. Alternately, a *Frankston Station* specification can report against the physical station component and detail both the rail services to be delivered and civil works required to upgrade *Frankston Station*.

Implementing inheritance and abstraction: Logical Architecture vs Physical Architecture

Logical architecture

Within the *Component* class we can create a design through a logical architecture that addresses the common functions performed by the logical components and describes the interfaces into each logical component. For the rail network, typical logical components could be the *Platform*, the *Rolling Stock* and the *Signalling System*, as illustrated in Figure 3.

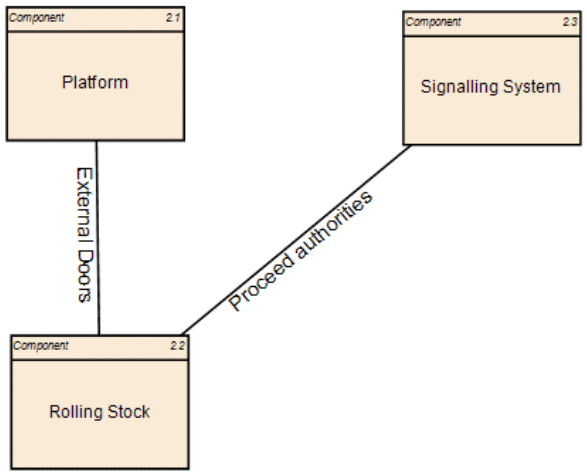

Figure 3 - Logical architecture

In this example, the *Platform* interfaces with the *Rolling Stock* by providing an external door interface. This allows the *Rolling Stock* to align correctly with the *Platform* to receive passengers. Similarly, the *Rolling Stock* has a signalling interface to receive and send proceed authorities.

Physical architecture

Within the *Component* class we can also create one or more *physical* architectures that describes the physical deployment concept for this rail network. We say that components in the physical architecture *instantiate* the logical components by inheriting an abstract parent component. Instantiation immediately imputes to each physical component the *Functions, Requirements* and *Interfaces* of the parent item. For the rail network the physical components are *Frankston Station*, *Prahan Station,* a *Siemens Train* and an *Automatic Train Control (ATC)* signalling system, as illustrated in Figure 4. Note how the inherited *Station* component is identified within the physical *Frankston Station* and *Prahan Station* components and the inherited interfaces are automatically included.

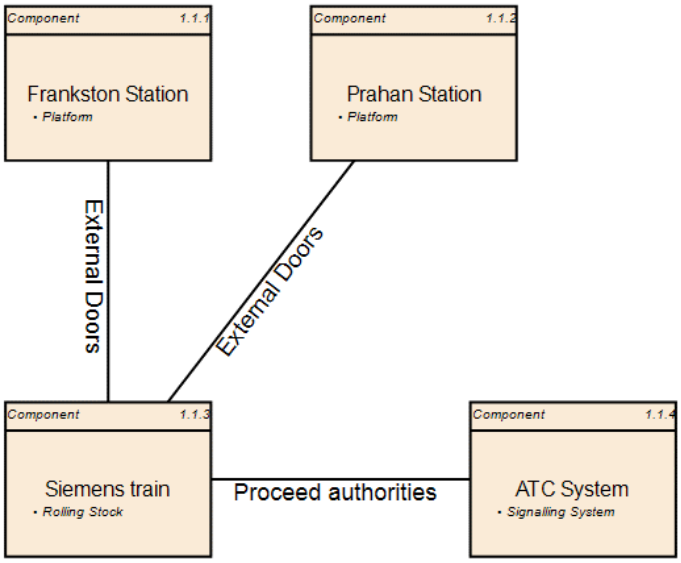

Figure 4 - Physical architecture

Implementing encapsulation

Within object-oritented systems engineering encapsulation is implemented through the logical architecture. During the logical design process, each logical component is allocated a set of *Functions*, and each *Function* is described by formal *System Requirements*.

When a physical component inherits the logical parent, it also inherits all of the allocated *Functions* and *System Requirements.* In this way, the engineer only needs to design each component once. In Figure 4 both *Frankston Station* and *Prahan Station* have identical behaviour because all of the behaviour characteristics are encapsulated within the logical *Platform* component. Engineers can instantiate multiple stations confident that the detailed behaviour in each station will be the same. The definition of those instantiations can exploit this explicit inherited structure to ensure that all inherited requirements are included in the design.

Inheritance and encapsulation can be extended by supplementing a physical component with additional site-specific functions, requirements or interfaces. For example, at *Frankston Station* we may decide to add a local communications link to the attached *Bus Interchange*, in order to coordinate operations. This is a local feature only, so we cannot add a new interface to the logical *Platform* component. Instead, we can add the interface directly to the *Frankston Station* site, as shown in Figure 5, without affecting the other station components.

A similar approach also supports tailoring of individual sites by adding specific requirements to individual sites whilst maintaining a consistent site functional description. An obvious example of site-specific requirements would be the civil works required, which are typically governed by local geography rather than system-wide functionality.

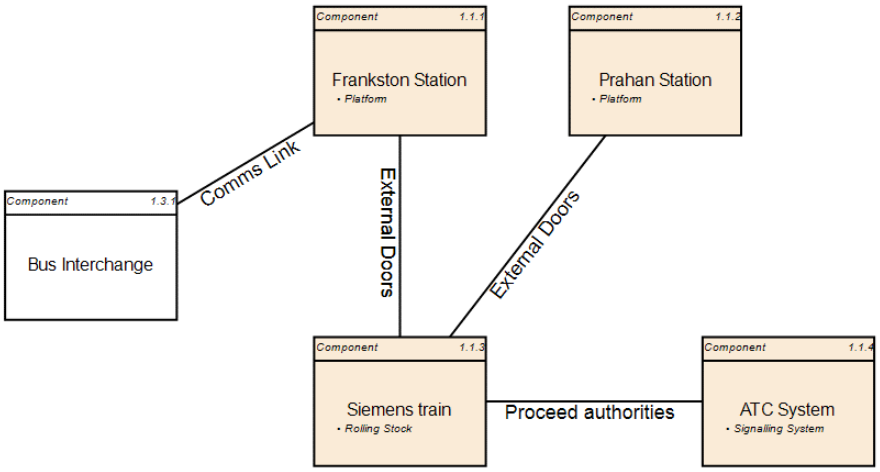

Figure 5 - Adding a local interface

Implementing model reuse

We have already seen how object-oriented systems engineering delivers model reuse through the ability to have a single logical component inherited by multiple physical components. However, model resuse can also be used to model alternate architectures.

Figure 6 illustrates an alternate architecture under consideration in which the signalling system is embedded at a specific railway station. In this architecture, *Frankston Station* inherits both a logical *Platform* and a logical *Signalling System*. Once again, the detailed functions and requirements do not have to be re-developed, since the logical design remains consistent. The logical *Navigation Signals* interface is automatically re-assigned between the train and the station, which illustrates how inheritance can be a powerful tool that supports quick development of alternate implementation options.

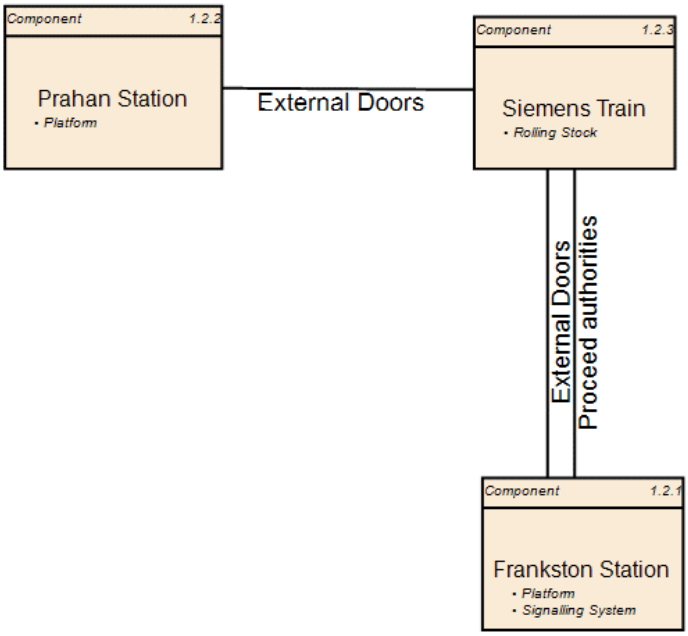

Figure 6 - Alternate architecture

CONCLUSION

MBSE is a proven design approach that encourages rigorous and systematic descriptions of complex system design. Within MBSE, the application of object-oriented design principles can enhance design

analysis on certain classes of project in ways that are difficult to replicate using traditional structured analysis.

Object-oriented systems engineering has been associated with software-intensive projects since the mid-2000s, primarily through the use of SysML. However, the design of infrastructure projects that require multiple physical nodes also provides a natural example that benefits from an object-oriented design approach. Key object-oriented concepts such as inheritance, encapsulation and component reuse provide an elegant way to analyse system functionality while also managing the physical design across multiple infrasturcture sites.

This paper has illustrated how object-oriented concepts can be applied to infrastructure projects without demanding that engineers learn the SysML language. Any reduction in the barriers to systems engineering will be welcome in industries where the application of systems engineering is less common.

REFERENCES

Bharathan, K, G.L. Poe and A.T. Bahill. "Object oriented systems engineering." *Proceedings of the 1995 International Symposium and Workshop on Systems Engineering of Computer-Based Systems*. Tuscon, AZ: IEEE, 6-9 March 1995. 69-76.

Estefan, Jeff A. "Survey of model-based systems engineering (MBSE) methodologies." *Incose MBSE Focus Group*. 2007. 1-12.

Friedenthal, S, A Moore and R Steiner. *A practical guide to SysML: The Systems Modelling Language*. 3rd Ed. Morgan Kaufmann/OMG, 2015.

Honour, Eric C. *Systems engineering return on investment*. Adelaide: University of South Australia, 2013.

INCOSE TP-2003-002-04. *INCOSE Systems Engineering Handbook; A Guide for System Life Cycle Processes and Activities*. Wiley, 2015.

Murray, Julia. *Model based systems engineering (mbse) media study*. International Council on Systems Engineering (INCOSE), 2012.

Pohl, Ira. *Object oriented programming using C++*. 2nd Ed. Addison Wesley, 1997.

Van Assche, F (ed.), Wieringa, RJ, Moulin, B (ed.) & Rolland, C (ed.). "Object-Oriented Analysis, Structured Analysis, and Jackson System Development." *University of Twente* (1991): 1-21.

Wymore, A. Wayne. *Model-Based Systems Engineering*. Boca Raton: CRC Press, 1993.

BIOGRAPHY

Dr Ian Brace is a systems engineer with experience in communications, military systems and public transport. His career has included military service, as well as extensive consulting to industry and government both in Australia and overseas. Dr Brace is an advocate for MBSE and has developed models for a broad range of projects including satellite systems; specialist vehicles; and rail communication systems. He also delivers training in MBSE and is the developer of Capability Architect, which supports model-based conceptual design and systems engineering.