Formal development and evaluation of narrow passageway system operations

Evangelos Kaisar [1]*, Mark Austin [2], Stratos Papadimitriou [3]

[1] Department of Civil Engineering, Florida Atlantic University, USA.

[2] Department of Civil and Environmental Engineering and Institute for Systems Research, University of Maryland, USA.

[3]Department of Maritime Studies, University of Piraeus, Piraeus, Greece

Abstract

This study applies a new intelligent transportation methodology for transforming informal operations concepts for narrow passageway systems into system-level designs, which will formal enough to support automated validation of anticipated component- and system-level behaviours. Models and specifications of behaviour are formally designed as labelled transition systems. Each object is the management system is assumed to have behaviour that can be defined by a finite state machine; thus, the waterway management system architecture is modelled as a network of communicating finite state machines. Architecture-level behaviours are validated using the Labelled Transition System Analyzer (LTSA). We exercise the methodology by working step by step through the synthesis and validation of a high-level behaviour model for a vessel passing through a waterway network (i.e., canal).

Keywords: Synthesis; Validation; Verification; Narrow Waterways Management; System Behaviour Model.

Introduction

Narrow passageway systems (e.g., waterway, work zone, tunnel, one-lane bridge and railroad applications) are large multidisciplinary complex systems characterized by geographically distributed system structures, concurrent subsystem-level behaviours, and end-to-end system life-cycles that may last decades. From a performance perspective, sophisticated techniques for engineering analysis are justified by adverse economics of poor system throughput. Within the waterways domain, for example, recent research has focused on assessment of overall system performance, congestion and delays in single and adjacent locks - see, for example, references (DeSalvo and

* Corresponding author: Evangelos Kaisar (ekaisar@fau.edu)

Lave, 1968; Dai and Schonfeld, 1998; Zhu et al. 1998; Ting and Schonfeld, 1998; Wang et al., 2006). Unfortunately, performance studies address only part of the problem - with traffic volumes expected to increase significantly into the foreseeable future (Austin and Kaisar, 2002; Maniccan, 2004), the effective management of passageways is needed to mitigate the undesirable impact of bottlenecks (perhaps caused by adverse weather conditions or accidents) on system safety, performance and cost.

As management systems become progressively diverse in their functionality, and solutions increasingly reliant on high technology, the challenge in creating good system-level designs will steadily increase unless new approaches are developed. In past decades, systems have been viewed from an operations point of view, where information and communication have been regarded as services necessary for the system to operate in pre-defined ways. Nowadays, there is a rapidly evolving trend toward large-scale information-dominated systems which exploit commercial off-the-shelf technologies and communications, have superior performance and reliability, and are derived in response to various types of information drawn from a wide array of sources (Austin, 2004). It is well known that concurrent behaviours increase complexity in scheduling activities to improve performance, while avoiding system failure. Nevertheless, there are now a growing number of application domains for which these difficulties are justified because of additional functionality that would not be possible without and ability to gather and work with data/information. For example, Turkey (Bosporus Straights), Korea (Tsushima Strait), and the Suez and Panama Canals have already made large investments to develop traffic management systems for narrow waterways where decision making is guided by GIS and data collected by land (sensors) centres (Paul, 1997; Gribar, 1999; Moore, 2000).

A fundamental tenet of our research is that new methodologies for system synthesis and validation are a prerequisite to software environments for front end system-level design of information-centric transportation management systems. For our purposes, synthesis of engineering systems is a process whereby provisional and plausible concepts are developed to the point where traditional engineering and design can begin. Synthesis is particularly important for systems that are quite unlike their predecessors; engineers need to return to first principles, rethink established ideas, and identify potential problems downstream in the development. The terms system validation and verification refer to two basic concerns, “are we building the right product?” and “are we building the product right?” Satisfactory answers to both questions are a prerequisite to customer acceptance.

To keep system developments on course and to prevent serious design flaws, today we seek validation and verification procedures that are an integral part of the development process (rather than a postscript to development) and support pre-deployment reasoning about system requirements and design. The complexity of design activities can be kept in check with system-level design methodologies that: (1) Strive to orthogonalise concerns (i.e., achieve separation of various aspects of design to allow more effective exploration of the space of potential design solutions) and (2) Employ formal design representation that enable early predictions of behaviour and detection of errors (Sangiovanni-Vincentelli, 2003). Moreover, to minimize the possibility of unforeseen failure we need models of system-level development that will help designers clearly articulate what the system must provide and what must be prevented. Engineers also need to understand the extent to which a system provides functionality beyond what is actually required.

Scope and objectives

Established approaches to behaviour modelling focus on the simulation of complete system descriptions (modelled to a certain degree of abstraction). Typical modelling objectives include performance assessment and identification of cause-and-effect relationships between system inputs and outputs (Cassandras, 1993; Fishwick, 1995). In a departure from this trend, this work focuses on the early stages of development where system descriptions may still be incomplete, but opportunities for improvement to the system design are greatest. Our primary goal is synthesis of high-level models of behaviour that can be used to verify correctness of partial system descriptions and guide the incremental elaboration of system descriptions. We note that while visual modelling languages, such as the Unified Modelling Language (UML, 2003) are useful for documentation and informal analysis, generally, they lack the precise interpretation of scenarios needed for rigorous analysis and formal verification of system compliance. A second problem is the failure of present-day techniques and tools to specify gaps in the specification which, if not detected, could result in costly design errors in the system development downstream.

Using ideas from object-based and systems-engineering development, this paper proposes a methodology for the incremental transformation of informal operations concepts into system-level designs, the latter being formal enough to support automated validation of anticipated component- and system-level behaviours. The methodology is inspired by our work on systems engineering methodologies (Austin, 2004; Kaisar et al., 2004) and, in part, the work of Magee and co-workers (Magee, 1999; Uchitel, 2004) on behaviour modelling for concurrent and distributed software systems.

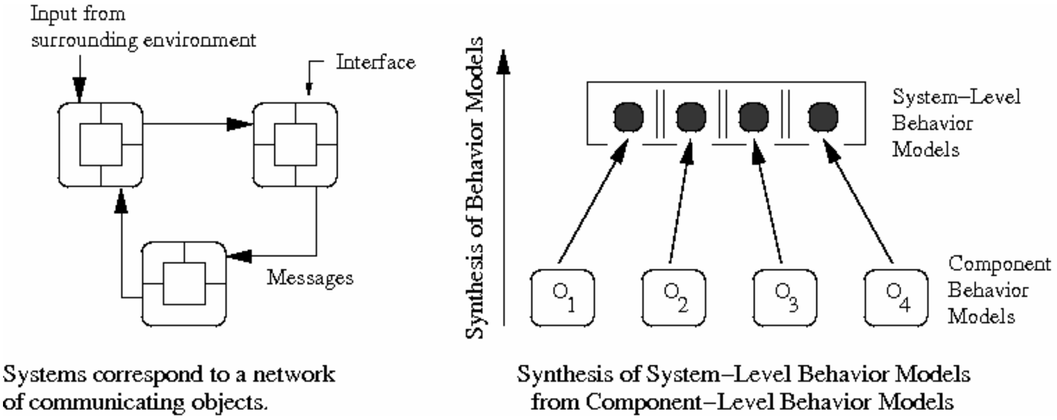
Input from surrounding environment
Interface
Messages
Systems correspond to a network of communicating objects.
Synthesis of Behavior Models
System–Level Behavior Models
Component Behavior Models
O_1
O_2
O_3
O_4
Synthesis of System–Level Behavior Models from Component–Level Behavior Models

Figure 1: Two Key Elements of Hybrid Object-Oriented/Systems-Development. Objects communicate through message passing (Source: Austin (2004)).

To exercise the methodology, we work step by step through the synthesis and validation of a high-level behaviour model for a ship passing through a waterway network. We assume that the waterway management system architecture can be modelled as a network of communicating objects, as shown on the left-hand side of Figure 1. Management systems achieve their purpose with object/module having well defined functionality, well defined interfaces for connectivity to other modules and the surrounding environment, and message passing. We assume that each object will have behaviour that can be defined by a finite state machine; thus, the management system will be modelled as an ensemble of interacting finite state machines. Scenario specifications and models of behaviour are formally modelled as labelled transition systems (LTSs) (Alur, 2000; Keller, 1976). At the component level, the nodes of a labelled transition system represent states the component can be in. At the architecture level, labelled transition system nodes represent system-level states, which, in turn, correspond to specific combinations of component-level states. Transitions are labelled with messages that components send to each other. The key advantage of this approach is that models of system-level behaviour can be automatically synthesized through the parallel composition of component-level behaviour models. A symbolic representation of this process is shown on the right-hand side of Figure 1. We validate behaviour using the Labelled Transition System Analyser (LTSA), a verification tool for concurrent systems (LTSA, 2004). In LTSA, processes correspond to sequences of actions. The textual representation is the finite state process (FSP) language. Labelled transition systems (LTSs) are the graphical representation. The properties required of the system are also modelled as state machines. LTSA mechanically checks that the specification of a concurrent system satisfies the properties required of its behaviour.

Frontend development

The methodology follows the step-by-step development procedure shown in Figure 2. Component- and architecture-level models of behaviour are synthesized using a combination of top-down and bottom-up strategies. A top-down decomposition strategy is used to develop component-level models of behaviour from use cases and scenario specifications. Models of architecture-level behaviour are developed through a bottom-up parallel composition of component-level behaviours. Downstream in the development (not covered by the methodology described here), system-level design alternatives are created by linking models of system-level behaviour to the high-level structure, and imposing constraints on performance and operation (e.g., control logic). To identify the major subsystems/objects and the details of message communication that a system-level design will need to support, we systematically work through the following steps: (1) Use cases and scenarios, (2) Basic- and high-Level message sequence charts, (3) System requirements, (4) Component- and architecture-level behaviours, an (5) Model checking and incremental improvement. The details of each step are as follows:

Use cases and scenario

Top-down development of system-level models begins with use cases, and proceeds to fragments of system functionality, expressed as activity and sequence diagrams (i.e., UML-based methods). Use cases are high-level representations of system functionality that do not reveal the details of implementation. Use case diagrams are a convenient way in which a real world actor (i.e., entities that are external to the system) will interact with the system, the use cases with which they are involved, and the boundary of the application. In a departure from established approaches to use case modelling

(Amour, 2001, Kulak 2000), the methodology organizes functionality according to aspects.

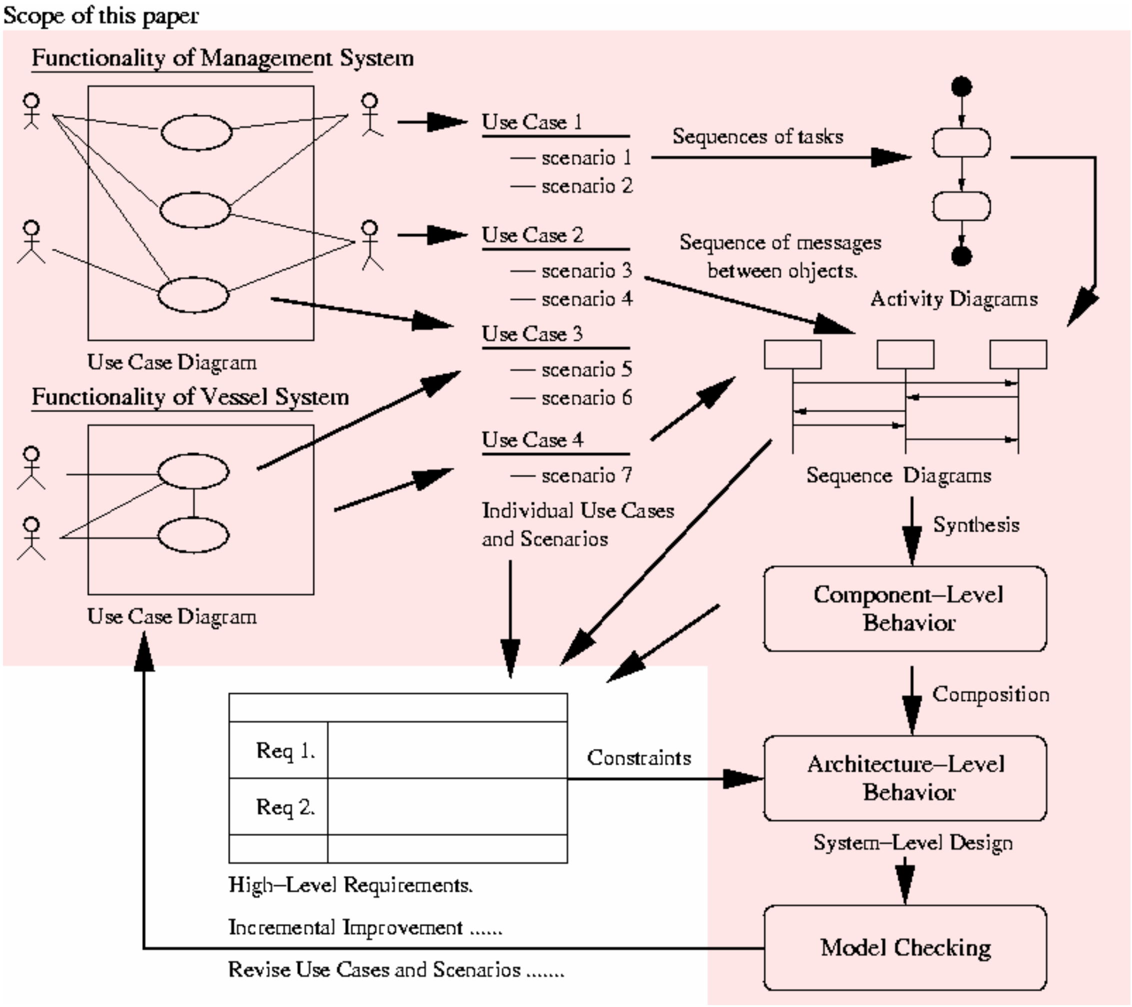

Figure 2: Step-by-step procedure for synthesis and validation of concurrent Object-Based Models for management of narrow passageways. Adapted from Austin (2004).

Aspects are viewpoints for handling concerns that cut across the details of implementation, particularly at the architecture level (Arnautovic, 2003; Jacobson, 2003). We employ aspects to represent the concurrent functionality of the main waterways management subsystems. The upper left-hand corner of Figure 2 shows, for example, functionality of the waterways system viewed from management- and vessel-system perspectives. Within each perspective, elements of system functionality can be partitioned into two parts: (1) Those concerned with operation of the viewpoint alone, and (2) Interactions between the management and vessel systems. Scenarios are partial descriptions of behaviour. Together they describe how the system components, the surrounding environment, and users interact in order to provide the system-level functionality. Several scenarios may be connected to each use case, explaining how the system will function under normal operations and alternative circumstances. Expecting stakeholders to produce a complete set of scenarios with complete coverage in one go is unrealistic.

To help designers articulate what must be provided by the system and what must be prevented, the methodology employs a combination of positive, negative and implied scenarios to guide incremental improvement of system-level descriptions (Uchitel, 2003; Uchitel, 2004). See Figure 3.The detailed model partitions scenarios into three categories. Positive scenarios specify the intended system behaviour. Negative scenarios specify undesirable behaviours the system is expected not to exhibit (e.g., operations that are unsafe). Implied scenarios correspond to gaps in the scenario-based specification. These gaps can occur in two ways. First, when models of architecture-level behaviour are composed from component-level behaviours, gaps in the scenario description will occur when individual component-level behaviour have an inadequate view of intended system-level behaviour. A second possibility is that the system architecture may contain feasible of traces of behaviour that are not detailed in the scenario specification (i.e., the system architecture might do something that the user is unaware of). An implied scenario may simply mean that an acceptable scenario has been overlooked and that the scenario specification need to be completed.

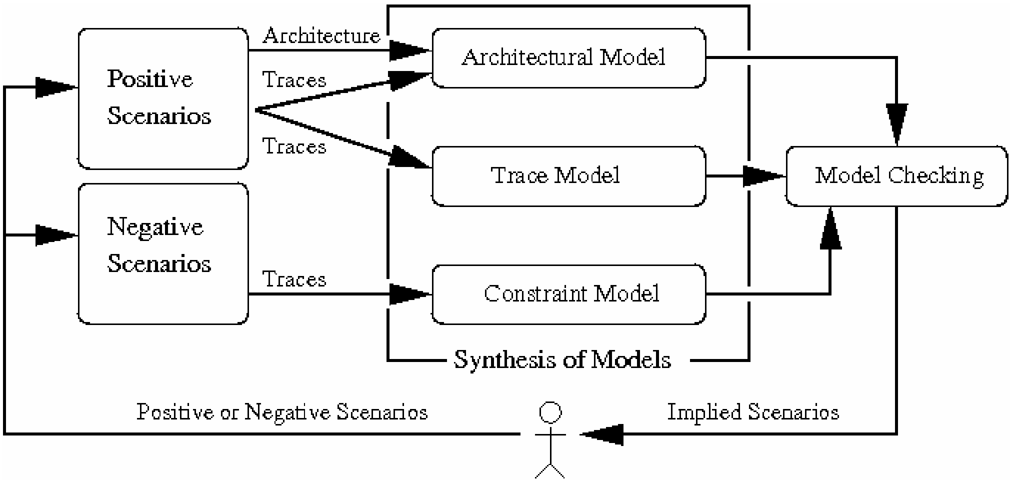

Figure 3: Flowchart for incremental synthesis of positive and negative scenarios, architecture trace and constraint models.

By detecting and validating implied scenarios it is possible to drive the elaboration of scenario-based specifications and behaviour models and possibly converge to a state where there are no more implied scenarios to be validated: (1) If a positive scenario is added as the result of accepting an implied scenario, then the specification for acceptable system behaviour is extended, (2) If a negative scenario is added as the result of rejecting an implied scenario, then the specification is strengthened. The decision to accept or reject a scenario depends on the problem domain at hand and will require consultation with the project stakeholders.

Basic and high-level message sequence charts

Figure 2 shows that activity and sequence diagrams can be derived directly from textual scenario descriptions. In UML nomenclature, activity diagrams show flows of task completion without revealing the details of internal implementation. Sequence diagrams show flows of communication among objects needed to implement system functionality (i.e., sequences of messages to complete a task), and in doing so, provide a high-level outline of the system architecture showing which components are involved in the implementation of fragments of behaviour.

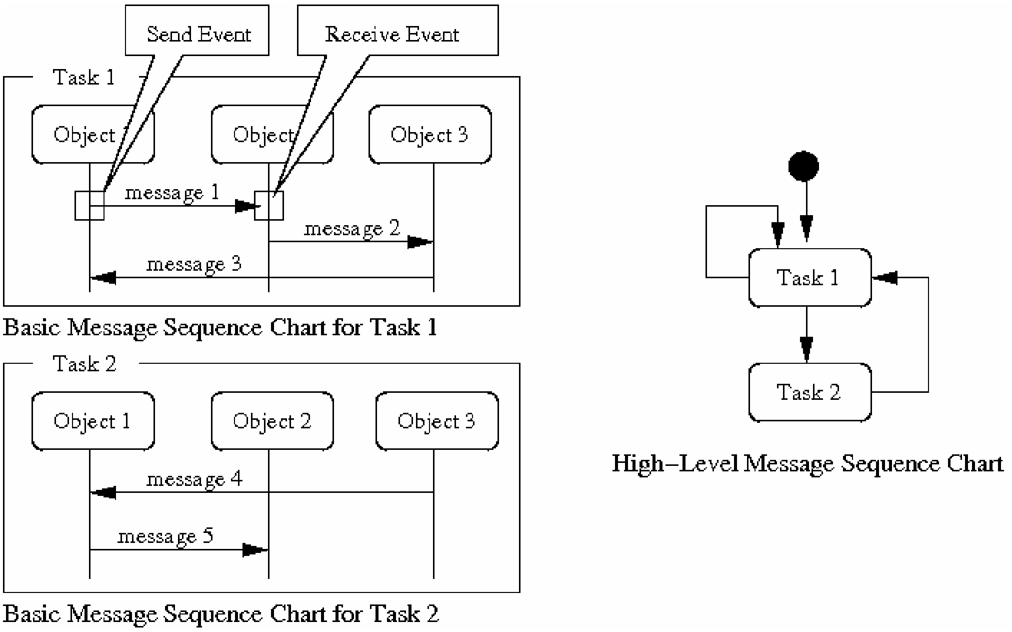

Figure 4: Elements of basic- and high-level message sequence charts.

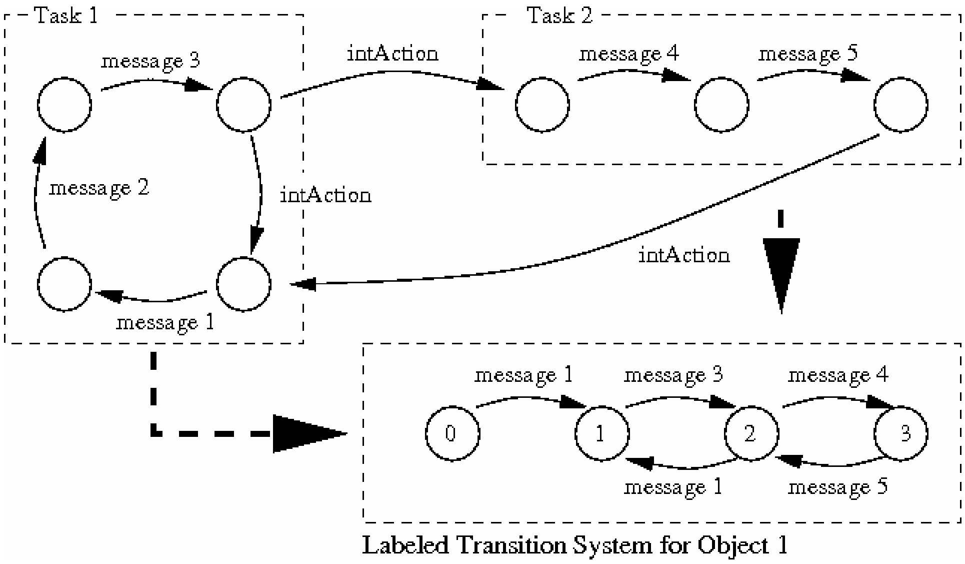

Figure 5: Synthesis of component-level model for object 1 from basic- and high-level sequence diagrams.

The methodology captures the dual benefit of sequence and activity diagram views by assuming that models of system-level behaviour correspond to a directed graph of nodes, with each node referencing either an individual task (i.e., sequence diagram) or a lower-level graph (i.e., activity diagram). For the purposes of behaviour model development, we adopt the notation of Alur (Alur, 2000) and Uchitel et al. (Uchitel, 2004), where basic- and high-level message sequence charts are used in lieu of sequence and activity diagram constructs, respectively. Basic message sequence charts depict sequences of messages (or traces) that components send to each other in order to complete a task. As illustrated in Figure 4, the vertical lines, called instances, are used to describe independent entities (e.g., Object 1, Object 2, Object 3). Messages represent interactions between the instances. Events correspond to observation of an interaction by the instances. Time in bMSCs is represented top-down -- that is, an event is considered to occur strictly after all the events that occur further up the instance timeline. We assume that message passing is synchronous; send and receive events are considered to occur simultaneously. This assumption keeps the analysis of behaviour models tractable.

The edges in high-level message sequence charts (hMSC) show how the system can evolve from once scenario to another, thereby allowing stakeholders to reuse scenarios within a specification and to introduce sequences, loops and alternatives to bMSCs. For example, behaviour of the hMSC in Figure 4 might correspond to one or more executions of Task 1, Task 2, one or more executions of Task 1, and so forth. Unfortunately, the interpretation of system-level behaviour is complicated by two possible interpretations of this evolution. One possibility is to assume that all components wait until all events in a bMSC have been completed before moving onto the next bMSC. As pointed out by Uchitel et al. (Uchitel, 2004) in order for this model to work, components need to know when a scenario has finished in order to the next to start - this implies components are, at a minimum, partially synchronized in their behaviour. The preferred approach, called weak sequential composition, assumes that components move into subsequent scenarios in an unsynchronized manner.

System requirements

Systems requirements correspond to constraints on system functionality, system interfaces, and non-functional concerns, such as safety and reliability. They are generated, in part, from features in the activity and sequence diagrams. For example, sequence diagrams also imply component interfaces needed to support the passing of message between components.

Component- and architecture-level behaviour

Models of component-level behaviour are built directly from the bMSC and hMSC specifications. Basic- and high-level specifications are translated in the form of finite sequential processes (FSPs), and graphically represented as labelled transition systems (LTSs). Briefly, the procedure for creating a component-level behaviour model is as follows: (1) Build one FSP process for each bMSC in the specification, (2) Build a process that models the behavior defined by the hMSC, (3) Define several auxiliary processes for each hMSC node. One process models the mapping of hMSC nodes to bMSC nodes. A second process is used to model continuations of the node according to connectivity relations in the hMSC. If concurrent scenarios have common elements, then there will be an interleaving of bMSC behaviours. Figure 5 shows, for example, the assembly procedure for creating a labelled transition system for Object 1. First, finite state process models are created for Task 1 and Task 2. The label "intAction" is used to specify connectivity of nodes in the hMSC. The last step in assembly of the component model is the make "intAction" unobservable and to minimize the model with respect to trace equivalence. Similar component-level models can be constructed for Objects 2 and 3.

Models of architecture-level behaviour are obtained through the parallel composition of concurrent processes at the component level. Given two labelled transition systems (LTSs) P_1 and P_2, we denote the parallel composition P_1 \| P_2 as the LTS that models their joint behaviour. By extension, the architectural-level behaviour model is defined by:

\begin{initial state}
{states that may be reached}
{From 0 we have the transition for P_1 and P_2}
\hbox {Architecture-Level Behaviour Model} = P_1 \| P_2 \| P_3 \cdots P_n
\end{Necessarily satisfied: equation}

where P_i is the finite state model for the i-th component among "n" interacting components. Joint behaviour is the result of all LTSs executing asynchronously, but synchronizing on all shared message labels. From an analysis perspective a good system: (1) exhibits safety and liveliness, and (2) avoids deadlocks. A safety property asserts that nothing bad will happen during the system execution. A liveliness property asserts that something good eventually happens (e.g., suppose that ships are approaching a narrow passageway. Liveliness would assert that, eventually, all of them will be able to pass through the passageway safely). A system state is deadlocked when there are no eligible actions that a system can perform.

Model checking and incremental improvement

Formal model checking procedures make sure that the architecture-level design: (1) does what it is supposed to do; (2) prevents certain behaviours from occurring; and (3) does not support un-intended behaviours. If any one of these aspects is violated, then we have a gap between the intended system and the actual system design. We can close gaps in the system design by refining the scenarios. This, in turn, leads to more detailed diagrams and a modified system-level architecture.

The analysis needs to include detection of traces that are exhibited by the architecture model, but have not been specified in the set of scenarios, and have not been explicitly rejected by stakeholders. The trace model is built from the set of positive system traces, ignoring the specified architecture (i.e., the components that make up the system architecture). The constraint model captures properties that the architecture model should comply with if it is to avoid the negative scenarios and provide only the specified behaviours. It is built from the set of negative traces - it captures the complement of traces the system should not exhibit.

Waterways application

In this section, we apply the methodology to the synthesis and validation of a simplified system-level behaviour model for a vessel passing through a narrow passageway, such as canals in the European inland waterway system. The behaviour model is assembled from two decoupled, but loosely interacting viewpoints: (1) Functionality of the vessel as it approaches and passes through the passageway, and (2) Functionality of the management system as it controls vessels passing through the narrow passageway. We assume that the management system will have at its disposal a variety of state-of-the-art technologies for collection and transmission of data (e.g., computers, cameras, GPS, sensors, and radio communications). Synthesis of the system-level behaviour model begins with a narrative description of system functionality. When the vessel is approaching the narrow point, the control centre will ask the vessel for information on its position. Initially, the vessel will be in a stand-by state. The driver of the vessel will send an inquiry to the control centre for transit availability, and remain in a stand-by state until it receives a command to proceed from the control centre. The control centre handles the request for transit availability by querying the control centre database for appropriate data/information, and any relevant events, such as accidents and delays. Eventually, the control centre will inform the driver on a decision about transit availability. Throughout this process, the vessel will send information on its position to the database. The database is automatically updated. In an alternative course of action, the control centre receives data from the database, and decides what it is going to be the next step for transit availability. Commands are sent to drivers of the appropriate vessels. Drivers will either move the vessel to the next control point and/or halt. When the transit procedure is complete, the vessel will send a terminate message to the control centre.

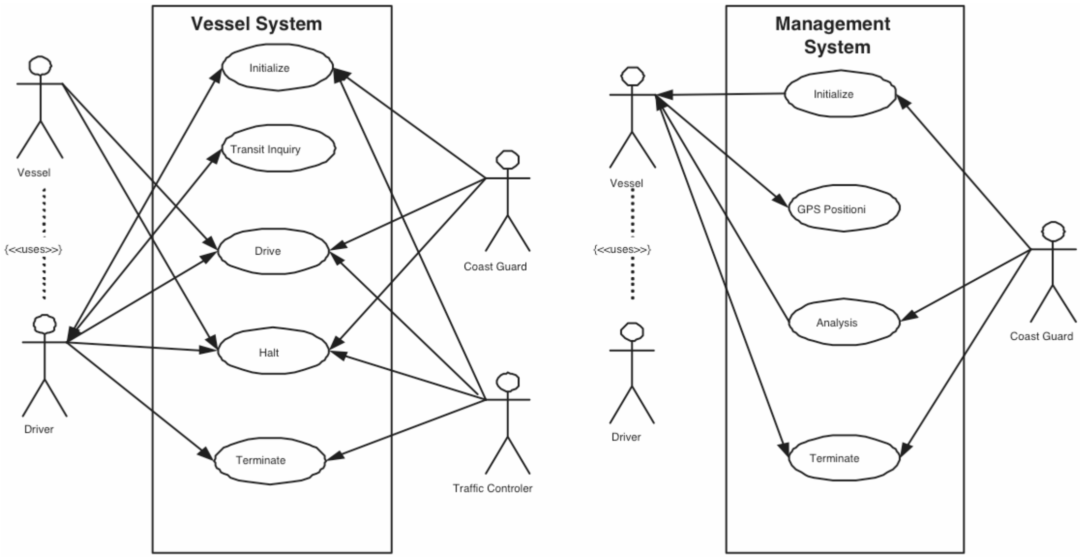

Figure 6: Use case diagrams for the vessel and Waterway Management Systems.

Use case model

Figure 6 shows high-level use case diagrams for the vessel system and a general purpose traffic management system. The names of the actors (which are drawn as stick figures) are Vessel, Driver, Traffic Controller, and Coast Guard At the heart of the traffic management system is the control centre. To maintain safety, ensure security and law enforcement, protect the environment, the control centre monitors weather conditions, tracks traffic in passageways, as well as scheduling and optimizing traffic operations. The major points of contact for a control centre are the traffic controllers, who implement the traffic policies imposed by the control centre. Controllers are physically located at the narrow passageway and can either be humans or automated devices. Points of contact also exist for the drivers/vessels who transit the narrow passageways. Drivers can register their presence with the control centre, and request guidance on traversal of the passageway. Periodically, the vessel will update the control centre on its geographical position.

This use case model provides a simplified view of functionality. A more realistic case study - details are not shown on Figure 6 - would include a transit-entry, transit-fullfilment, and channel advisory subsystems, supported by a geographic information system (GIS). The transit-entry subsystem would have use cases for looking up transit availability of the waterway, creating a new transit request, and updating the transit schedule. Transit-fulfillment takes care of scheduling and traffic control policies. Channel advisory subsystems send updates on channel conditions to operators, controllers, drivers and the coast guard. These additional features contribute to the safe and efficient use of large-scale waterway systems.

Scenarios

Functionality of each use case can be elaborated into detailed scenarios represented in textual and graphical formats. As a case in point, from the left-hand side of Figure 6 the Transit Inquiry use case expands to:

```
Primary Actor(s): Driver.
    Description
   The control centre analyzes the driver's inquiry for transit.
  Pre-conditions:
       The onboard technology of transiting vessels includes
       GPS receivers and telecommunication equipments.
    Flow of Events:
       1.The driver uses the onboard equipment to send an inquiry
          to the control centre.
       2.The control centre receives the inquiry.
       3.The control centre queries the management database for
          most up-to-date information.
       4.Information is sent from the database to the control centre.
       5.The control centre resolves the "transit request"
          and sends the decision result to the driver.
    Post-conditions:
       Information about the transit is updated.
       The control centre returns to standby mode.
    Alternative flow of Events:
       None.
    Assumptions:
       Waterway is in use.
```

Textual scenarios provide an effective way of eliciting system-level objects that will be involved in the execution of the scenario. For the Transit Inquiry use case, the driver will interact with a control centre, which, in turn, will communicate with a database.

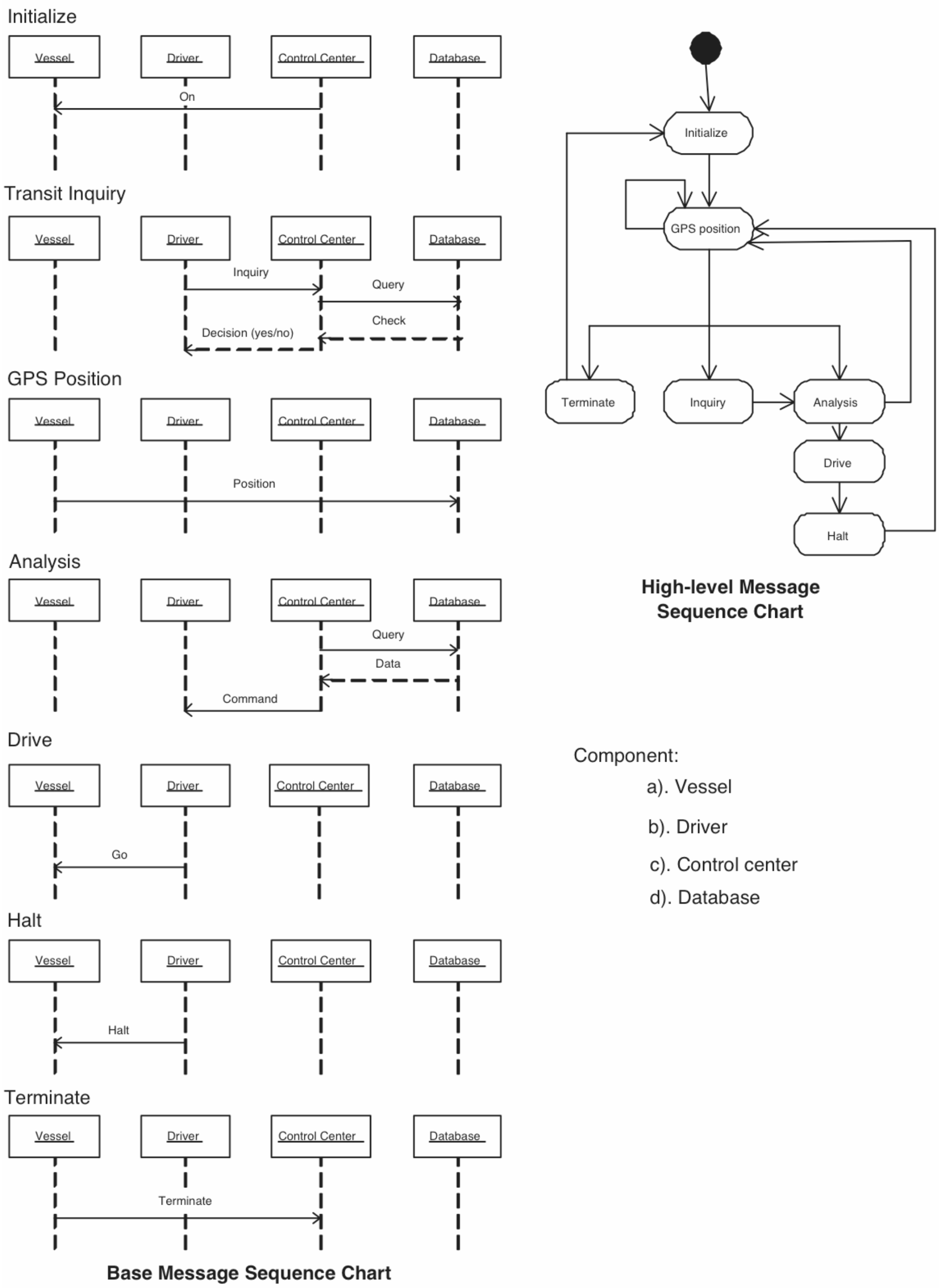

Figure 7: Basic- and high-level message sequence chart (MSC) specifications for waterway system functionality.

Sequence and activity diagrams

As a prerequisite to modelling system-level behaviour in LTSA, we create a two-level message sequence chart (MSC) for a graphs of tasks that need to be completed when a vessel passes through the passageway. System-level behaviour corresponds to a directed graph of tasks. The left-hand side of Figure 7 shows sequences of messages passed among four entities - the vessel, the driver, a control centre, and a database - for execution of the tasks Initialize, Transit Inquiry, GPS Position, Analysis, Drive, Hall, and Terminate. Each box and vertical line in the sequence diagram corresponds to an object in the system. Horizontal arrows correspond to messages passed between the objects, which will need to be part of the system structure. The right-hand side of Figure 7 shows a directed graph of tasks, from which system-level behaviour emerges. Notice that the vessel and drivers are actors in the use case model. The control centre and database are components in the management system. The GPS system sends a continuous stream of data to the database. The control centre unit performs decisions and send commands to the vessel and driver. The database receives queries from the control centre and returns appropriate data/information.

Component- and system-level architectures

Finite state models of component- and architectural-level behaviour are composed directly from the message sequence chart. First, finite state representations for component behaviour are obtained using the procedure outlined earlier in this paper.

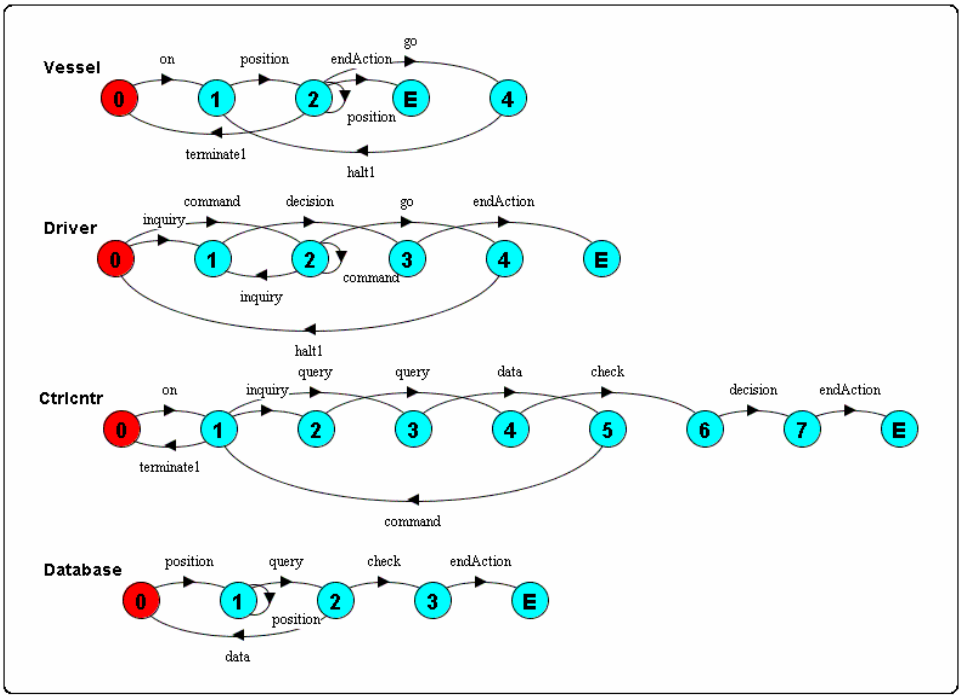

Figure 8: Component models in waterways application.

Figure 8 shows the behaviour of each component (such as vessel, driver, GPS position, and database) is exactly the same as described in the activity diagram. Models of system-level behaviour (also termed architecture-level behaviour) are obtained through a parallel composition of component finite state machines. The architecture model has 43 nodes, and is too large to be shown here. The trace model has 15 states and it is shown in Figure 9. At the component level, the numbered nodes in the graphical representation correspond to permissible states. At the architecture level, the numbered nodes correspond to specific combinations of states at the component level.

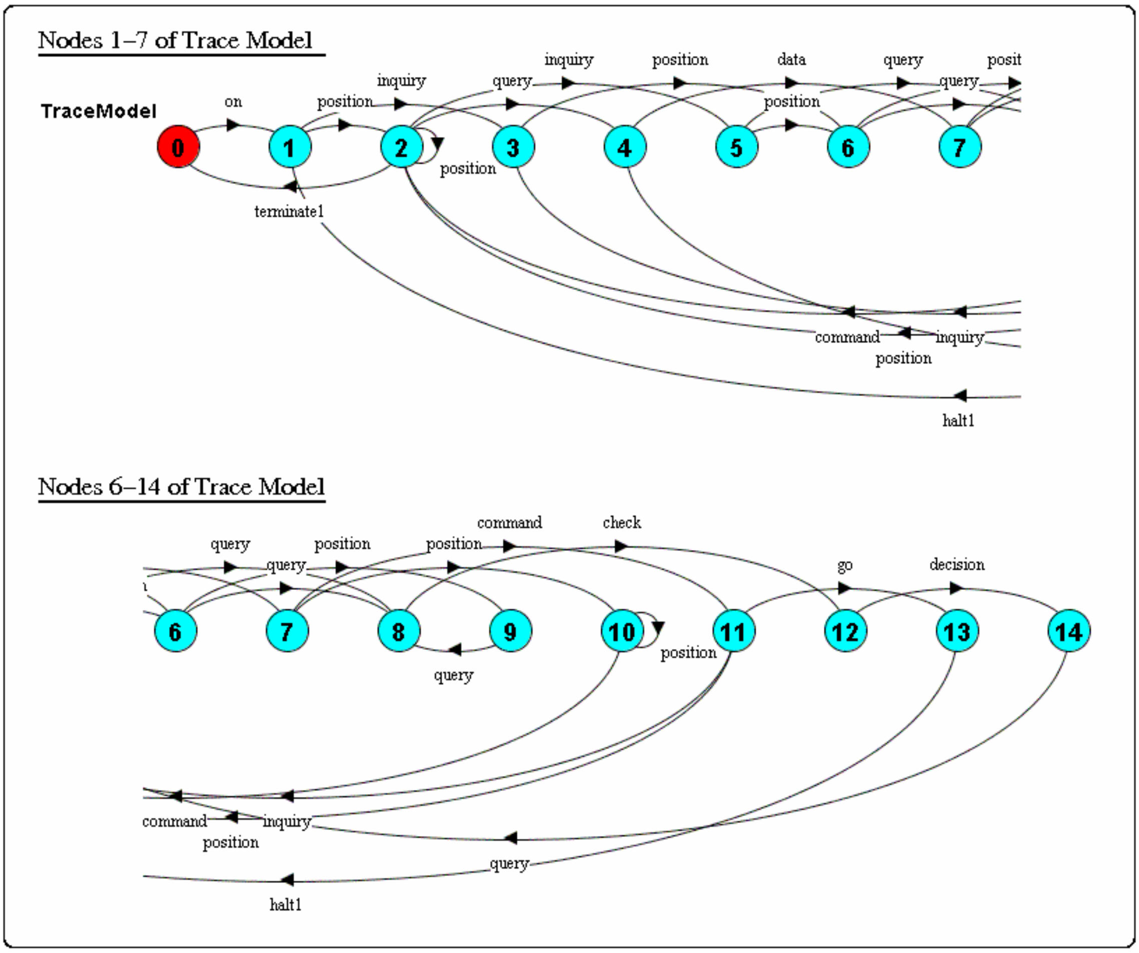

Figure 9: Trace model in waterways application.

Implied scenarios

An implied scenario is a sequence of message labels that appear as a trace prefix in the architecture model, but are neither a prefix of positive- nor negative- traces. Figure 10 shows, for example, a trace (the basic sequence chart syntax for clarity) that can be executed in our system model, but hasn't been explicitly classified as being a positive or negative scenario. By looking back at Figure 7 we see that the message sequence on, position, terminate1, on, and query corresponds to the task sequence Initialize, GPS Position, Terminate, Initialize and Analysis. From the textual description of behaviour it is clear that Terminate is intended to only occur after the ship has traversed the passageway. Hence, we classify this scenario as undesirable behaviour -- it is registered as a negative scenario and added as a constraint in the model checking procedure. Formally, this negative scenario has a basic format, consisting of the message sequence

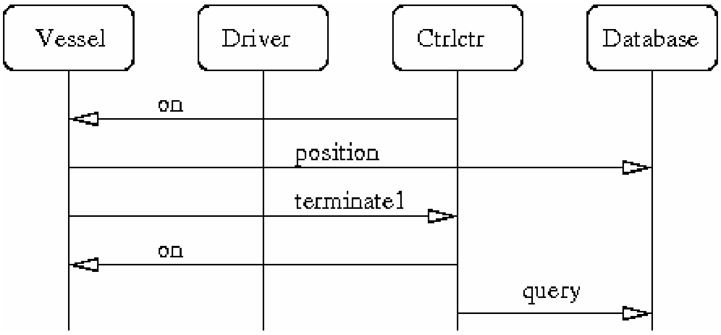

Figure 10: Implied scenario in model of waterways behaviour.

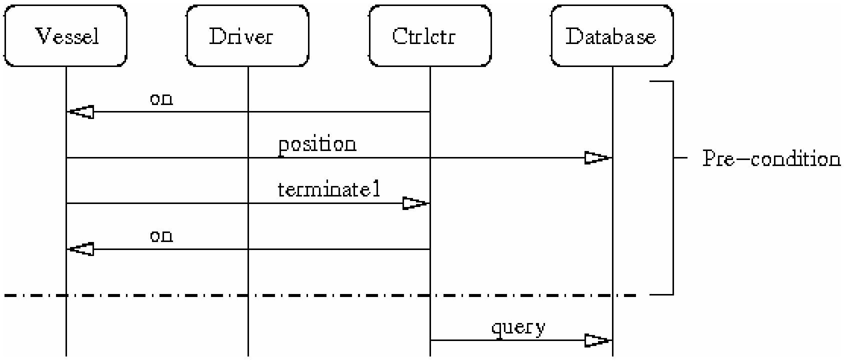

Figure 11: Representation of basic negative scenario.

(p,l), where "p" is a basic MSC for the pre-condition and "l" is the proscribed (prohibited) message. See Figure 11. Uchitel and co-workers have developed an ensemble of progressively expressive formats for describing various types of negative scenarios.

Conclusions and future work

In this paper we have demonstrated how ideas from object-based development and systems engineering can be combined to create a methodology for the incremental transformation of informal operations concepts (represented by use cases and positive, negative and implied scenarios) into formal representations for system-level behaviour (represented by labelled transition systems). The latter representation is formal enough to support automated evaluation of operational and safety concerns. These concerns complement engineering estimates of system performance. However, the methodology presented in this paper is simply a first step.

Looking ahead, as waterway management systems become more complex and reliant on high-technology (software and hardware), we anticipate a strong need emerging for formal procedures that can help engineers move in a systematic manner from use cases and scenarios to requirements, to architecture-level (logical) representations of a system, followed by the detailed (physical) design and testing. Appropriate forms of validation/verification need to be weaved into each step of this development process. One issue this paper did not address is testing. It should be possible, in principle, to use the architecture-level (logical) specification of a system to guide the generation of test suites needed for the lower level physical design. In the present investigation, refinements to the use-case and scenario models are driven by discrepancies in traces through the system-level architecture that are not supported by the message-sequence chart specification. Due to the large number of components in a transportation system and, in part, the uncertainty in human behaviour, devising a complete list of working scenarios can be very difficult. To mitigate this shortcoming, there is a need to explore the use of ``waterways simulators" for the generation of scenarios (Mahmassani, 2004). Finally, nowhere in the model do we talk about operations whose correctness depends on time. To overcome this limitation, we are currently exploring the potential benefits of timed automata and systems analysis/verification of concurrent behaviours with UPPAAL (UPPAAL, 2004).

References

Alur, R., Etessami, K., and Yannakakis, M. (2000) "Inference of Message Sequence Charts", *International Conference on Software Engineering (ICSE'00)*, Limerick, Ireland.

Armour, F. and Miller, G. (2001) *Advanced Use Case Modelling,* Addison-Wesley, New York.

Arnautovic, E. and Kaindl, H. (2003) "Aspects for Crosscutting Concerns in System Architectures", In *CSER: Conference on Systems Engineering Research*, University of Southern California, Los Angeles, April.

Austin, M. A. and Kaisar, E. (2002) "Multi-Level Analysis of Congested Transportation Systems", *6th world Multiconference on Systemics, Cybernetics and Informatics*, July 14-18, Orlando, Florida.

Austin, M. A. (2004) "An Introduction to Information-Centric Systems Engineering", Tutorial F06, INCOSE, Toulouse, France, June.

Cassandras, C. G. (1993) *Discrete Event Systems: Modeling and Performance Analysis*, Irwin and Aksen Associates.

Dai, M. D. and Schonfeld, P. (1998) "Metamodels for Estimating Waterway Delays Through a Series of Queues", *Transportation Research*, Vol. 32, No. 1, pp. 1-19.

DeSalvo, J. S. and Lave, L. B. (1968) "An Analysis of Towboat Delays", *Journal of Transportation Economic Policy*, pp. 232-241.

Fishwick, P. A. (1995) *Simulation Model Design and Execution: Building Digital Worlds*, Prentice-Hall, New York, New York.

Gribar, J. C., and Bocanegro, J. A. (1999) "Passage to 2000 (Modernization of the Panama Canal)", *Civil Engineering Magazine*, December.

Jacobson, I. (2003) "Use Cases and Aspects Working Seamlessly Together", *Journal of Object Technology*, Vol. 2, No. 4, pp. 7-28.

Kaisar, E., Austin, M. A. and Haghani, A. (2004) "An Object-Oriented Approach for the Architecture Design of the Management of Narrow Passageways", In *International Workshop on Harbour, Maritime and Multinodal Logistics, Modelling and Simulation*, Copacabana, Rio de Janeiro, Brazil, September 16-18.

Keller, R. (1976) "Formal Verification of Parallel Programs", *Communications of the ACM*, Vol. 19, No. 7, pp. 371-384.

Kulak, D. and Guiney, E. (2000) "Use Cases: Requirements in Context", *Addison-Wesley*, New York, New York.

Magee, J. L. and Kramer, J. (1999) *Concurrency: State Models and Java Programs*, John Wiley and Sons.

Magee, J. L., Kramer, J. and Uchitel, S. (2004) "Labelled Transition System Analyzer (LTSA) Home Page." See:http://www.doc.ic.ac.uk/~jnm/book/ltsa/LTSA.html.

Mahmassani, H. (2004) "Personal Communication", Department of Civil and Environmental Engineering, University of Maryland, College Park, November.

Maniccan, S. (2004) "Congestion of point-to-point mobile objects," *Physica A: Statistical Mechanics and its Applications*, Volume 331, Issues 3-4, 15, January, pp. 669-681.

Moore, M. (2000) "The Bosporus: A Clogged Artery", *The Washington Post*, 16th November.

Paul, L. (1997) "Aressel Traffic System Analysis for Korea/Tsustima Strait", *Energy-Related Marine Issues in the Regional Seas of Northeast Asia*, Berkeley, CA, USA.

Sangiovanni-Vincentelli, A. (2003) "Electronic-System Design in the Automobile Industry", *IEEE Computer Society*, pp. 8-18, May-June.

Ting, C. J. and Schonfeld, P. (1998) "Integrated Control For Series of Waterway Locks", *Journal of Waterway, Port, Coastal, and Ocean Engineering*, ASCE, 124(4), pp. 199-206.

Uchitel, S. (2003) "Incremental Elaboration of Scenario-Based Specifications and Behavior Models using Implied Scenarios", *Ph.D. Thesis*, Imperial College, London, England.

Uchitel, S., Kramer, J., and Magee, J. (2004) "Incremental Elaboration of Scenario-Based Specifications and Behaviour using Implied Scenarios", *ACM Transactions on Software Engineering and Methodology*, Vol. 13, No. 1, pp. 37-85, January.

Unified Modelling Language (UML) (2003). See http://www.omg.org/uml.

UPPAAL: An integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata. (2004). See http://www.uppaal.com/.

Wang, S., Tao, X. and Schonfeld, P. (2006) "Shippers Response to Scheduled Waterway Lock Closures", *85th Annual Transportation Research Board Meeting*, Washington DC.

Zhu, L., Schonfeld, P., Kim, Y., Flood, I. and Ting, C. J. (1998) "Queuing Network Analysis for Waterways with Artificial Neural Networks", *Artificial intelligence for Engineering Design, Analysis and Manufacturing*, pp. 365-275.