# *Formal Methods for a System of Systems Analysis Framework Applied to Traffic Management*

Charles E. Dickerson, Siyuan Ji, Rosmira Roslan
The Wolfson School. Mechanical, Electrical and Manufacturing Engineering.
Loughborough University
Loughborough, UK
{c.dickerson, s.ji, r.roslan}@lboro.ac.uk

***Abstract*— Formal methods for systems and system of systems engineering (SoSE) can bring precision to architecting and design, and increased trustworthiness in verification; but they require the use of formal languages that are not broadly comprehensible to the various stakeholders. The evolution of Model Based Systems Engineering (MBSE) using the Systems Modeling Language (SysML) lies in a middle ground between legacy document-based SoSE and formal methods. SysML is a graphical language but not a formal language. Initiatives in the Object Management Group (OMG), such as the development of the Foundational Unified Modeling Language (fUML) seek to bring precise semantics to object-oriented modeling languages. Following the philosophy of fUML, we offer a framework for associating precise semantics with Unified Modeling Language (UML) and SysML models essential for SoSE architecting and design. Straightforward methods are prescribed to develop the essential models and to create semantic transformations between them. Matrix representations can be used to perform analyses that are concordant with the system of UML or SysML models that represent the system or SoS. The framework and methods developed in this paper are applied to a Traffic Management system of systems (TMSoS) that has been a subject of research presented at previous IEEE SoSE conferences.**

***Keywords—cyber physical system; formal methods; graphical modeling language; safety critical; semantic transformation***

## I. INTRODUCTION

Safety critical systems and system of systems (SoS) can be found in a variety of commercial domains such as aerospace and commercial nuclear facilities, to name just two. Failures in these types of systems can result in injury or even death. The increasing use of embedded systems and controllers to cope with the growing complexity of systems underscores the importance of understanding the behavioral aspect of systems and SoS. Safety aspects are normally governed by international safety standards for systems and software. For example, compliance with DO-178C, Software Considerations in Airborne Systems and Equipment Certification [1], is recognized as an acceptable means for verifying airworthiness and certification of the software aspects of airborne systems.

DO-178C, in particular, provides guidance regarding the use of model-based development, object-oriented techniques, and formal methods. However, Model Based Systems Engineering (MBSE), when implemented using the Object Management Group Systems Modeling Language (SysML), cannot be regarded as formal because SysML is not a formal language. The OMG has been long been pursuing precise executable languages. In 2008, the adoption of the Foundational UML (fUML) provided the first precise operational and base semantics for a subset of UML for object-oriented activity modeling [2-4]. The semantics defined by fUML specify a virtual machine for executing models compliant to this subset [2]. However, practical viability could not be realized until the fUML Action Language (Alf) was developed and adopted. The success is attributable to the philosophy that Alf notation can be attached to a UML model any place that a behavior can be. The base semantics play an important role in the formal verification of UML models [3].

Further precision can be brought to modeling through semantic transformation between models. This can make the process of model based system definition both more precise and traceable, if not more repeatable. Although the term semantic transformation is frequently used in the literature, e.g. in domains such as the semantic web, it is often used without offering a precise definition. Furthermore, across different domains the term can have different meanings.

Chu et al [5] have regarded semantic transformation as the replacement of specific data with contextual information. Such transformation of data can add semantic knowledge from different aspects, e.g. adding a street name to a GPS spatio-temporal sample.

We shall use the term in a precise way that is based on the Klir general systems methodology for scientific problem solving [6]. The methodology is concerned with the interpretation of an abstraction of a system into a model of the system, e.g. a model of a problem to be solved that is interpreted into the features of the system that demonstrate a solution to the problem. The interpretation provides new information that is not in the abstraction.

Related to the Klir methodology is the concept of viewpoint (e.g. in [7]). A viewpoint on a system is a

This work was partially sponsored by the Programme for Simulation Innovation (PSi). A partnership between Jaguar Land Rover and UK EPSRC grant EP/K014226/1.

technique for abstraction using a selected set of architectural concepts and structuring rules to focus on particular concerns within the system. (See also [8].) By the term abstraction is meant a process of suppressing selected to establish a simplified model. When the Klir methodology is combined with the IEEE concept of viewpoint, a precise definition of semantic transformation can be offered:

*Semantic transformation is a technique for the interpretation of an abstraction of (or related to) a system into a semantically richer model by using a specified set of modeling and structuring rules.*

This is depicted graphically in Fig. 1; it both includes and generalizes the replacement concept of transformation used by Chu. The rules establish what type of new information is to be added to the abstraction and how the information will be organized to create the system model. A similar but formal definition of ontological transformation is offered in [9].

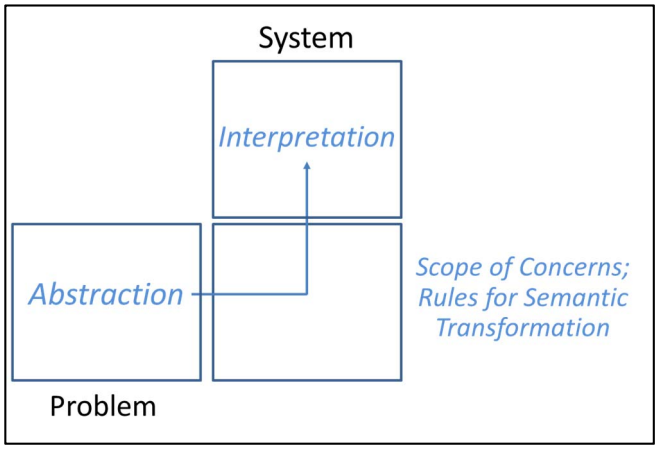


Fig 1. The Klir methodology as a framework for sematic transformation

The abstraction can also be a graphical model in an object oriented language; with the interpretation of the abstraction being a matrix. When the graph of an object oriented model is interpreted as adjacency matrix, precision is achieved because the matrix has exactly the same entities and relationships as the graph.

Using the definition of semantic transformation offered above and following the philosophy of fUML, it will be possible to create a chain of UML or SysML models linked by semantic transformation that can be represented by matrices. This representation will provide precise semantics that can enable direct computation in the analysis of safety critical cyber physical systems and SoS.

The purpose of this approach is to interpret statements in object oriented languages such as UML and SysML into mathematical matrix representations which are formal. Traditional formal methods include model checking, formal testing, and proving of theorems. When object oriented UML or SysML models have been rendered in matrix form, a variety of other formal methods can be exploited. Some of these are discussed in the next section.

## II. Matrix Methodology

The methodology developed in this paper is concerned with matrix representations of SysML graphical models. The method augments the traditional adjacency matrix of mathematical graph theory and includes control structures. The use of an adjacency matrix resembles a dependency structure matrix (DSM) in identifying dependencies, information exchanges, or interactions via matrix elements. However, a matrix associated with semantic transformation is not an adjacency matrix; rather, it associates nodes between graphs.

When the matrix representation is created using semantic transformation, the matrices can be used in several ways. These methods include: efficient test case generation, Activity insertion, Activity grouping, entry and loop detection; and counting and identifying fault paths in Activity diagrams.

These methods for using the matrix representation are explained in more detail as follows:

- Test Case Generation: the matrix representation can allow the efficient generation of test case by tracing the non-zero matrix elements from row to column. Algorithms can be constructed to automatically trace through matrix elements to build up test cases.
- Activity Insertion: the matrix representation can also provide a systematic method for implementing new Activities into the system without reading out complex Activity Diagrams.
- Activity Grouping: as demonstrated in [10], grouping of Activities based on sequential order and the nature of the Activities allows simplification of Activity Diagrams. Further, for failure state testing, instead of testing through individual Activities, one can instead perform at the Grouping level. Then, by identifying the Activity Group in which the failure cause resides, one can expand the Group to allocate the root failure Activity. The matrix representation allows efficient grouping of Activities though matrix element tracing.
- Entry and Loop Detection: efficient algorithms have been developed to detect entry and loop in adjacency matrices [11-12]. These methods can be used to identify intended and non-intended loops in complex Activity Diagram
- Finally and importantly, the adjacency matrix supports formal methods for prediction and identification of root cause of system failures at the design stage using (graphical) walks through the matrix. Counting and identifying walks in graphs is just one useful method from the mathematical theory of graphs [13].

The scope of our paper will focus on the later point and the semantic transformations that can be used to create the matrix representation of object oriented models and demonstrated in the case study (Section V).

## III. A Framework for Semantic Transformation

The behavioral modeling of a system or SoS in UML or SysML can be accomplished through seven steps linked by semantic transformation. Specifically,

1. Determine what the SoS is supposed to do [to achieve its stated purpose(s)].
2. Elaborate what the SoS is supposed to do into a high level but complete set of actions.
3. Organize the flow and control of the actions.
4. List the combination of systems that are proposed to comprise the SoS; then associate (allocate) each action with one or more or the systems.
5. Organize the systems by defining the intended interactions needed to accomplish allocated tasks.
6. Govern the interactions through the order of exchanges and structure of control.
7. Specify interfaces to enable collaboration.

The scope of this paper will be limited to first three steps and the analysis limited to safety critical behavior modeling. These will be sufficient to address the TMSoS Case Study in Sections IV, V. The essential UML models will be identified for the first three steps, semantic transformations specified, and matrix notation defined.

**Step 1**: Use Case Definition. The Scope of Concern is the Elements of the Environment (Actors) and how the SoS is used (Use Cases) based on interactions.

*Rule for Semantic Transformation:*

Actors and high level Use Cases are associated by interactions.

*Matrix Representation:*

1. List of Actors: $C_1^a, C_2^a, \dots, C_k^a$ , where $a$ denotes Actor;
2. List of high level Use Cases: $A_1^0, A_2^0, \dots, A_l^0$;
3. The Sematic Transformation Matrix is formed of intended parings:

$\boldsymbol{Q_1} = [(C_i^a, A_j^0)]$ with matrix elements: $Q_{i,j} = 1$ if $C_i^a$ is associated with $A_j^0$; $Q_{i,j} = 0$ otherwise.

**Step 2**: Use Case Elaboration. The Scope of Concern is the high level Use Cases and how they are elaborated into Activities by means of UML extensions and inclusions.

*Rules for Semantic Transformation:*

1. Inclusion: for each high level Use Case model, elaborate any 'included' Activities needed to complete the model;
2. Extension: for each high level Use Case model, elaborate any 'extension' Activities needed to complete the model;

*Matrix Representation:*

1. List of high level Use Cases: $A_1^0, A_2^0, \dots, A_l^0$;
2. List of elaborated Activities: $A_1^s, A_2^s, \dots, A_m^s$, where $s$ denotes System;
3. The Sematic Transformation Matrix is formed by intended the inclusions and extensions:

$\boldsymbol{Q_2} = [(A_i^0, A_j^s)]$ with matrix elements: $Q_{i,j} = 1$ if $A_i^0$ includes $A_j^s$; $Q_{i,j} = -1$ if $A_i^0$ is extended by $A_j^s$; and $Q_{i,j} = 0$ if $A_i^0$ does not have direct association with $A_j^s$.

It should be noted that between steps 2 and 3, each Use Case is then expressed in a table as a list of activities, with conditions and extension points. As this is not a graphical model, these details are not shown.

**Step 3**: Creation of the Activity Diagram. The Scope of Concern is the elaborated Activities, the Activities associated with the Elements of the Environment, and organization of them into an Activity Diagram by means of flows and controls.

*Rules for Semantic Transformation:*

Adjacent Activities or an Activity and control node are associated by directed edges.

*Matrix Representation:*

1. List of all Activities in the Activity Diagram:
   - System Activities: $A_1^s, A_2^s, \dots, A_m^s$;
   - Environment Activities $A_1^E, A_2^E, \dots, A_n^E$;
   - Control Nodes: $A_1^C, A_2^C, \dots, A_p^C$;
   - Entry/Exit point counts as Activities in the corresponding swim lane in the Activity Diagram;
2. The Sematic Transformation Matrix (*directed adjacency matrix*, i.e. non-symmetric) is formed by intended directed connections.

$\boldsymbol{Q_3} = [(A_i^X, A_j^X)]$ with matrix elements: $Q_{i,j} = 1$ if $A_i^X$ flows into $A_j^X$; $Q_{i,j} = 1$ if $A_i^C$ flows into $A_j^X$ with a "Yes" result, and $Q_{i,j} = -1$ if $A_i^C$ flows into $A_j^X$ with a "No" result; and $Q_{i,j} = 0$ otherwise.

Note that the transformation semantics specified in each of the above steps are precise: in every graphical model (diagram), each node and line in the diagram is uniquely associated with a cell in an adjacency matrix.

## IV. TMSoS Case Study Review

The case study in this paper is based on [14], which was presented at the IEEE SoSE 2014 conference. The paper used the Fault Modeling Architecture Framework (FMAF) to provide a systematic approach to capture fault tolerance

aspects of SoS. The FMAF study was concerned with a TMSoS that controls the inter-urban road network in Netherlands. It was carried out through collaboration between a research group from Newcastle University and West Consulting. The study concluded that quality tools and methods are needed to support reasoning about SoS at the architectural level for recovery strategies which affect SoS service quality. Extended FMAF views in SysML were produced to support reasoning of degraded level of SoS service and the representation of failures. The fault tolerance functions of a Ramp Meter System (RMS) situated on the access inter-urban highway was described using FMAF.

By using the Fault Tolerant Structure View (FTSV), redundancy of similar services and effects from constituent systems (CSs) were identified. From the identified redundancy, negative and positive influences of CSs towards the SoS goal permit architectural engineers to reason faults/failures at the SoS level. Exactly five SoS-level faults were cited that could arise within the RMS. The faults were then stored in a Fault/Error/Failure View diagram. These will be the subject of our matrix based analysis.

## V. Application to the Case Study

In this section, we apply the developed framework to formally architect the RMS based on the following requirements of the TMSoS and Traffic Control Centers (TCC) derived from [14]:

- The RMS is *situated on the access ramp* used to access inter-urban highways.
- The RMS **employs two-phase (red and green) traffic lights** to **control the rate** at which vehicles join the highway.
- The RMS *prevents bottlenecks* from being formed when many vehicles join a major road, *improves vehicle distribution* by breaking up platoons, and can *reduce accidents caused by high speed merges*.
- An RMS typically *has access to data about traffic* in its own immediate vicinity (as opposed to the TCC which has access to region wide traffic data)
- The RMS operates in one of several modes to include:
  1. a *fixed time mode*, with fixed length red/green phases;
  2. an *adaptive Mode*, which responds to current traffic conditions;
  3. a *collaborative Mode*, where TCC **overrides** the RMS decision.
- RMS may be **gathering local data and making decisions** in isolation, or **receiving instructions** from the TCC instead.
- The RMS regularly **collects and analyses data** about the local area and, when necessary, **selects a responsive mode** (adaptive or collaborative) of operation

In the narrative, the System of interest is the RMS. The actors (in red) are underlined, the functional requirements (in blue) are bold, and non-functional requirements (in orange) are italic.

According to the framework for semantic transformation, the first step in architecting the system is to identify the actors in the environment and system use cases based on interactions. This is straightforward: the system (the RMS) interacts with Vehicle and TCC to achieve the high level Use Case, "control Vehicle Flow".

**Step 1**. To capture the high level Use Case in matrix representation, the following lists are defined and variable names are given accordingly:

*List of Actors:*
$C_1^a \equiv$ Vehicle
$C_2^a \equiv$ TCC

*List of high level Use Cases:*
$A_1^0 \equiv$ control Vehicle Flow

The two lists and the rule for semantic transformation in Step 1 are used to construct the first matrix,

$$\mathbf{Q}_1 = \begin{matrix} \\ C_1^a \\ C_2^a \end{matrix} \begin{matrix} A_1^0 \\ \end{matrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

**Step 2.** In the second step, the use case is elaborated based on the detailed functional requirements. This step includes identifying the relationships among the sub-level use cases including inclusion and extension. The detailed model of the elaborated Use Case is depicted in Fig. 2.

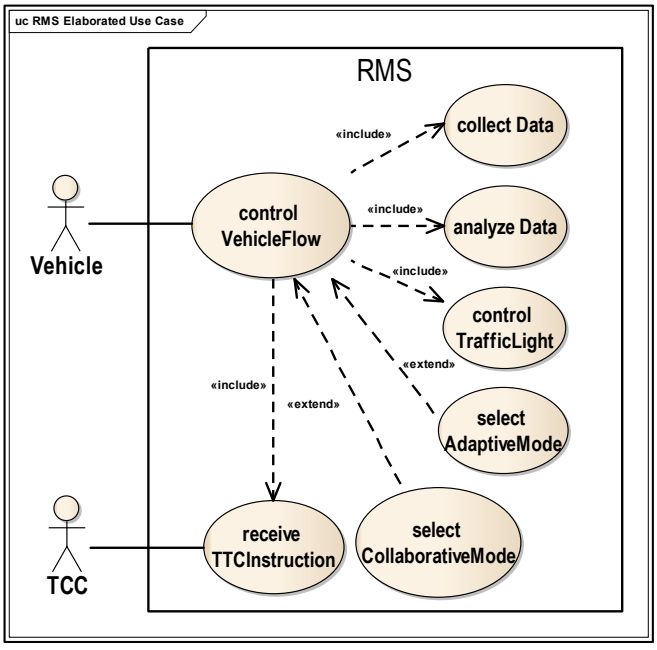


Fig 2. Elaborated Use Case for RMS

To construct the matrix representation of the graphical model, the following list of activities is created.

*List of elaborated Activities*
$A_1^s \equiv$ collect Data
$A_2^s \equiv$ analyze Data
$A_3^s \equiv$ receive TTC Instruction
$A_4^s \equiv$ select Adaptive Mode
$A_5^s \equiv$ select Collaborative Mode
$A_6^s \equiv$ control Traffic Light

Using the rules for semantic transformation, the matrix reads,

$$\mathbf{Q}_2 = A_1^0 \begin{array}{c} \begin{matrix} A_1^s & A_2^s & A_3^s & A_4^s & A_5^s & A_6^s \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 & -1 & -1 & 1 \end{pmatrix} \end{array}$$

**Step 3.** In this step, the elaborated use cases are organized into an Activity Diagram, as depicted in Fig. 3. Apart from the system Activities, to have a complete behavior description, Activities of the Actors in association with the system Activities are also identified.

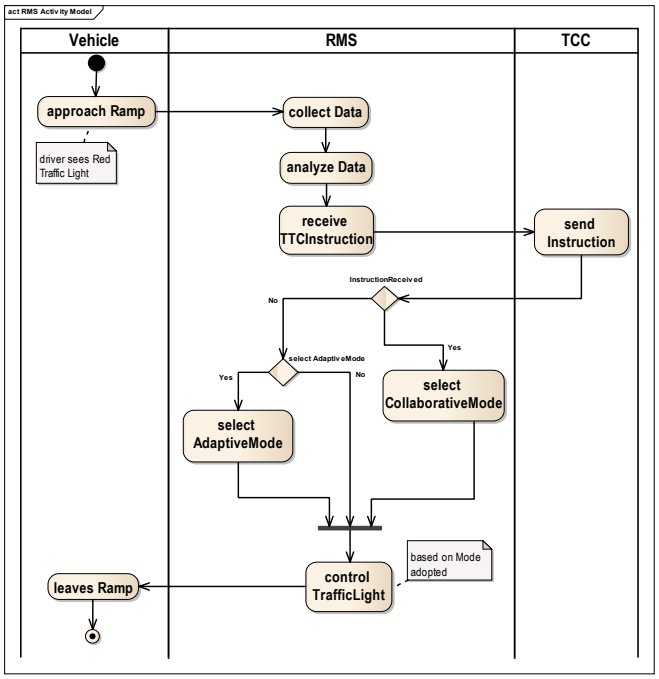


Fig 3. Activity diagram

Taking the Activities of the Actors and control nodes into consideration, the complete list of activities is:

*List of all Activities*
$A_1^v \equiv$ point of entry
$A_2^v \equiv$ approach Ramp
$A_3^v \equiv$ leave Ramp
$A_4^v \equiv$ point of exist

$A_1^s \equiv$ collect Data
$A_2^s \equiv$ analyze Data
$A_3^s \equiv$ receive TTC Instruction
$A_4^s \equiv$ select Adaptive Mode
$A_5^s \equiv$ select Collaborative Mode
$A_6^s \equiv$ control Traffic Light

$A_1^c \equiv$ Control Node: instruction recieved
$A_2^c \equiv$ Control Node: switch to Adaptive Mode

$A_1^T \equiv$ send Instruction

where the superscripts $v$ denotes vehicle, and $T$ denotes TCC. The Activities with these superscripts belong to the environment but are represented in different swim lanes. Based on the directed connection between Activities, the matrix representation of the Activity Diagram is depicted in Fig. 4. The representation provides an efficient approach for allocation of faults to associated Activities, as well as prior Activities that might happen to be the root cause of the system faults. The combination of the behavioral modeling and the matrix representation exposes faults more readily than a component based model.

$$\mathbf{Q}_3 = \begin{array}{c} \begin{matrix} A_1^v & A_2^v & A_3^v & A_4^v & A_1^s & A_2^s & A_3^s & A_4^s & A_5^s & A_6^s & A_1^c & A_2^c & A_1^T \end{matrix} \\ \begin{matrix} A_1^v \\ A_2^v \\ A_3^v \\ A_4^v \\ A_1^s \\ A_2^s \\ A_3^s \\ A_4^s \\ A_5^s \\ A_6^s \\ A_1^c \\ A_2^c \\ A_1^T \end{matrix} \begin{pmatrix} & 1 & & & & & & & & & & & \\ & & & & 1 & & & & & & & & \\ & & & 1 & & & & & & & & & \\ & & & & & & & & & & & & \\ & & & & & 1 & & & & & & & \\ & & & & & & 1 & & & & & & \\ & & & & & & & & & & & & 1 \\ & & & & & & & & 1 & & & & \\ & & & & & & & & 1 & & & & \\ & & 1 & & & & & & & & & & \\ & & & & & & & & & & 1 & & -1 \\ & & & & & & & 1 & & -1 & & & \\ & & & & & & & & & & & & \end{pmatrix} \end{array}$$

Fig 4. Matrix represnetation of the Activity Diagram in Fig. 3

As noted in Section IV, reference [14] cited exactly five SoS-level system faults that could arise within the RMS:

1. *Lights stuck on green or no lights at all*
2. *RMS fails to adopt collaborative mode when instructed*
3. *Lights stuck on red*
4. *RMS fails to exit collaborative mode when instructed*
5. *The RMS calculates an incorrect rate for vehicles to be admitted*

These faults will be seen to map to a larger subset of behavioral faults that are mathematically derivable from the matrix representation of the Activity diagram.

The adjacency matrix in Fig. 4 admits three (graphical) walks from the initial node (collect Data) to the final node (control TrafficLight). These are the basis for tracing root causes to system level behavioral faults. Counting and identifying walks in graphs is mathematically well understood; and can be used to organize discussions such as follows.

Fault numbers 1 and 3 are directly associated to Activity, $A_6^s$, i.e. control Traffic Light. If having tested that the fault does not resides in the Traffic Light itself, one needs to trace prior Activities that provide input to $A_6^s$. Tracing $A_6^s$ form the Activity Matrix, $\boldsymbol{Q_3}$, one can quickly identify three Activities

prior to $A_6^s$, which are $A_4^s$, $A_5^s$, and $A_2^c$. Testing these Activities and further up (if necessary) allows the identification of the root cause of the failure. An example root cause could be an error in the Collaborative Mode strategy that leads to an indefinite green light or red light implementation for the RMS.

Fault 2 and 4 are directly associated with Activities $A_5^s$ and $A_1^c$. Here, a root cause for these two failures can be RMS failing to receive instruction from TCC such that the Control Activity, $A_1^c$, cannot be executed.

Fault 5 is associated with $A_1^s$ and $A_2^s$, where a possible root cause could be RMS failing to detect any vehicle, hence leading to an incorrect vehicle rate calculation.

In summary, the matrix representation allows identification of potential system level faults at the design stage. The incorrect execution of an activity (including control nodes) will clearly be associated with one or more system faults. Hence, each activity forms a root cause of one or more CS faults. Since the connectivity of the Activities are reflected by non-zero matrix elements in $\boldsymbol{Q_3}$, these non-zero elements can then be used to trace root cause of the system faults.

## VI. Conclusions and Future Work

Using semantic transformations, we have offered a framework for developing UML and SysML models essential for SoSE architecting and design. This was the basis for a process of model based system definition that was seen to be both more precise and traceable, if not repeatable. Following the philosophy of fUML and using the association matrices such an adjacency matrix of the graphical models, we have elaborated the framework to associate precise semantics with UML and SysML models. This permitted the use of straightforward methods use the matrix representation to perform mathematical analyses that were concordant with the UML and SysML models that represent the system or SoS.

When applied to the TMSoS case study that had been presented at the IEEE SoSE 2014 conference [14], the matrix representation directly identified three behavioural faults that were associated to the five SoS level system faults reported by [14] within a key constituent system (the RMS). Five further behavioral faults were also noted using the matrix representation. A combination of behavioral modeling with the association to the constituent systems of the TMSoS provides a more complete picture of the SoS fault states.

The proposed next step of the framework would be to construct a semantic transformation matrix that allocates Activities as Operations that owned by system components (Classes or Blocks). The association matrix for semantically transforming Activities Diagram to Class or Block Diagrams is expected to provide insights into the consistency of Sequence Diagram construction. Moreover, it is anticipated that this association matrix further allows the identification of potential system failures that resides within individual system component through the traceability of the matrix.

When the Activity Diagram, Sequence Diagram and transformation between them are represented in matrix form, mathematically based consistency checks can then be made using previous work of the author [15-16].

## References


[1] DO-178C RTCA/DO-178C: Software Considerations in Airborne Systems and Equipment Certification, Dec 2011

[2] T. Mayerhofer, P. Langer, M. Wimmer, and G. Kappel, "xMOF: Executable DSMLs Based on fUML," in Proc. of the 6th International Conference Software Language Engineering (SLE), Indianapolis, USA, pp. 56-75, Oct 2013

[3] A. G. Romero, K. Schneider, and M. G. V. Ferreira, "Using the Base Semantics given by fUML for Verification," in Proc. of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), Lisbon, Portugal, pp. 5-16, Jan 2014

[4] L. Berardinelli, P. Langer, and T. Mayerhofer, "Combining fUML and Profiles for Non-Functional Analysis Based on Model Execution Traces," in Proc. of the 9th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA), Vancouver, Canada, pp. 79-88, June 2013

[5] D. Chu, et al., "Visualizing hidden uhemes of taxi movement with semantic transformation," in Proc. of the 7th IEEE Pacific Visualization Symposium (PacificVis), Yokohama, Kanagawa, Japan, pp.137-144, Mar 2014

[6] G. J. Klir, Facets of Systems Science, New York, U.S.: Plenum Press, 1991

[7] IEEE Recommended Practice for Architectural Description for Software-Intensive Systems, IEEE Standards 1471-2000, 2000

[8] ISO/IEC/IEEE Systems and Software Engineering-Architecture Description, ISO/IEC/IEEE Standard 42010, 2011

[9] Y. D. Wang, H. Ghenniwa, and W. Shen, "Ontological view based semantic transformation for distributed systems," in Proc. of the IEEE International Coference on Systems, Man, and Cybernetics (SMC), San Antonio, Texas, USA, pp. 4007-4012, Oct 2009

[10] D. Ewen, G. Pai, and I. Whiteside, "Formal foundations for hierarchical saftey cases," in Proc. of the 16th IEEE International Symposium on High Assurance Systems Engineering, (HASE), Daytona Beach, Florida, USA., pp. 52-59, Jan 2015

[11] R. Sedgewick and K. Wayne, Algorithms, 4th ed., Massachusetts: Addison-Wesley Professional, 2011

[12] R. Yang, R. Gao, and C. Zhang, "A new algebraic approach to finding all simple paths and cycles in undirected graphs," in Proc. of the IEEE International Conference on Information and Automation (ICIA), Lijiang, Yunnan, China, pp.1887-1892, Aug 2015

[13] L. W. Beineke and R. J. Wilson (Eds), Topics in Algebraic Graph Theory, Cambridge, U. K.: Cambridge University Press, 2004

[14] C. Ingram, Z. Andrews, R. Payne, and N. Plat, "SysML fault modelling in a traffice management system of systems," in Proc. of the 9th Interantioanl Conference on System of Systems Engineering (SOSE), Adelaide, Australia, pp. 124-129, June 2014

[15] C. Dickerson, "A Brief History of Models and Model Based Systems Engineering and the Case for Relational Orientation", IEEE Systems Journal, Vol. 7 No. 4 (2013), pp. 581-592

[16] C. Dickerson, "A Relational Oriented Approach to System of Systems Assessment of Alternatives for Data Link Interoperability", IEEE Systems Journal, Vol. 7 No. 4 (2013), pp. 549 – 560