A novel graph-based modelling approach for reducing complexity in model-based systems engineering environment

Maxim Filimonov*, Prof Ilias Oraifige, Dr Venkatesh Vijay

Birmingham City University, Faculty of Computing, Engineering and the Built Environment, Millennium Point, Curzon Street, Birmingham, United Kingdom, B4 7XG

Email: maxim.filimonov@mail.bcu.ac.uk

Email: ilias.oraifige@bcu.ac.uk

Email: venkatesh.vijay@bcu.ac.uk

*Corresponding author

Abstract

Field of systems engineering is developing rapidly and becoming more complex, where multiple issues arise like overcomplexity, lack of communication or understanding of the design process. Model-based systems engineering (MBSE) has been introduced to overcome these issues and reduce systems complexity. Nonetheless, the system model remains static and the interactions among submodels are modelled in the form of hard-coded rules. Therefore, systems interaction in MBSE is not dynamic enough to satisfy the evolving nature of system models with growing complexity.

In this paper, a novel approach for modelling logic is proposed to address the above challenge. The aim of the research is to improve existing methodologies and interaction modelling as well as deal with inconsistencies. The approach is based on graph theory, where pre-defined rules and relationships are substituted and reorganised dynamically with graphical constructs while metagraphs being the most applicable for systems modelling. Framework for reducing complexity is presented.

Keywords: model-based systems engineering, graph-based modelling, object-oriented approach, metagraph, systems engineering, design structure matrix, systems modelling language, design engineering

Biographical notes: Maxim Filimonov is a postgraduate research student doing a PhD degree in Engineering at Birmingham City University in the UK. He got his Master of Science degree in 2013 at Moscow Institute of Physics and Technology in Aerospace Research while being involved in the process of aerodynamics calculations of the new spacecraft. Maxim Filimonov has experience in working in Aerospace Industry where he worked for 3.5 years as a Senior Design Engineer at the main Russian enterprise for developing airships. His current area of interest lies within the fields of Model-Based Systems Engineering and Knowledge Based Engineering. More specific, he is interested in automation and making the development of complex systems with multiple interacting components easier for systems engineers.

Professor Ilias A. Oraifige is a highly qualified chartered engineer and a fellow member of the Institution of Mechanical Engineers (IMechE). He has extensive research track record and successful industrial experience, possesses good communication skills, energy and thorough knowledge of new technology. Currently he is working as Professor and Head of Centre of Engineering at Faculty of Computing, Engineering and the Built Environment at Birmingham City University. The role requires management of 27+ fulltime staff, associate lecturers and technical staff plus over 650 students. He is solely responsible for the budget and its allocation including sessional lecturers, resources, etc.

Dr Venkatesh Chennam Vijay received his PhD in Engineering for extending Knowledge-Based Engineering (KBE) principles into engineering distance learning from Birmingham City University (BCU). Currently he is working as a Lecturer in the Electronic Engineering at Birmingham City University. At present Dr Vijay is the module leader and has been delivering mathematic lecture and workshop for Foundation students and further has being developing and module-coordinating Knowledge Based Engineering (KBE) module for Automotive/Mechanical Masters students. Dr Vijay has developed various engineering educational demos including human robot collaboration, creative design space and an Augmented Reality (AR) based teaching and learning environment.

1 Introduction

In the modern era, a lot of different definitions of engineering has been derived. Engineering itself is “the creative exploitation of energy, materials and information in organized systems of people, machine, and environment systems which are useful in terms of contemporary human values” (Wymore, 1993). At the same time, systems engineering (SE) arouses an advanced way of engineering by providing an inter-disciplinary approach and means to enable the realisation of successful systems from different points of view. SE is “an inter-disciplinary approach and means to enable the realisation of successful systems” (Haskins, 2006). It has been evolving rapidly in the past decades and the rate of this evolvement has risen dramatically recently. This leads to the growing complexity of the systems being designed with the use of SE methods as more sophisticated and larger systems are being developed nowadays. Numerous issues have become problematic for successful system development process, which are overcomplexity, lack of communication and lack of understanding of the design process at different stages of a lifecycle (Holt and Perry, 2008).

Model-based systems engineering (MBSE) is an emerging approach in the SE field that distinguishes itself as an advanced way to reduce and maintain systems complexity through storing all development knowledge in an organised model structure. As a fundamental principle of good system design, the essence of MBSE relies on the application of appropriate formal

models to a given domain (Bahill and Botta, 2008). MBSE itself is "a formalised application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later lifecycle phases" ("Systems Engineering Vision 2020," 2007). Despite the rapid development of the MBSE field significant issues exist in the way of its further development. These issues involve overcomplexity, lack of understanding and proper interaction among different models as they are parts of the main system model.

Main system model in MBSE is decomposed into multiple submodels corresponding to separate subsystems (Yassine and Braha, 2003). These submodels represent various aspects of the development process - design engineering, computational analysis, cost model, manufacturing analysis, requirements model, etc. All the components and systems are in constant interaction among each other but this interaction is not modelled in a way to automatically and dynamically update the system on time as well as check the consistency of the development process on its every stage (Shekar et al., 2011). The interactions within systems need to be understood and identified before they can be modelled using MBSE methodologies. One of the ways to analyse systems is using the decomposition principle, which is one of the key aspects of engineering helping to organise complex problem in the initial stages of the systems development.

Once the model interactions are understood and the communication such as messages, decisions, and responses among models are set to be analysed. In this research, such communication among systems, subsystems and components is defined as logic. In current systems this logic is maintained manually through hard-coded rules, pre-defined relationships, constraints and fixed mathematical formulas (Estefan, 2008). This increases the time required for the actual development and further leads to designing the system from scratch whenever serious and contradicting problems are discovered at the later stages of the lifecycle. Moreover, often there are logical contradictions – inconsistencies in rules and dependencies among the rules that are not captured (Herzig et al., 2014). This could lead to incorrect system design, increased time spent on testing, redesign, and ultimately systems failures.

Holt and Perry discuss the broader issue facing SE and call them "three evils of systems engineering" (Holt and Perry, 2008). These issues have been distinguished as follows:

- Complexity: large systems have lots of interacting components and relationships between its entities. As shown in Figure 1, adding more rules and relationships make the system more complex than before.
- Lack of understanding: the concept of "lack of understanding" can arise at any stage of a lifecycle beginning with requirements formulation. It might lead to issues during the

development stage, and then even during the operation of a product. Understanding of a system model is a major factor in any development (Friedenthal et al., 2014). Models have to be dynamic as the systems are usually observed only from particular aspects of a certain design. Similarly, it is legit for overcomplicated models with lots of communication, rules and relationships.

- Communication problems: This problem can arise on any level, between several people or groups of people, companies, systems or different departments involved in process of development, where there is not enough data exchange among various submodels.

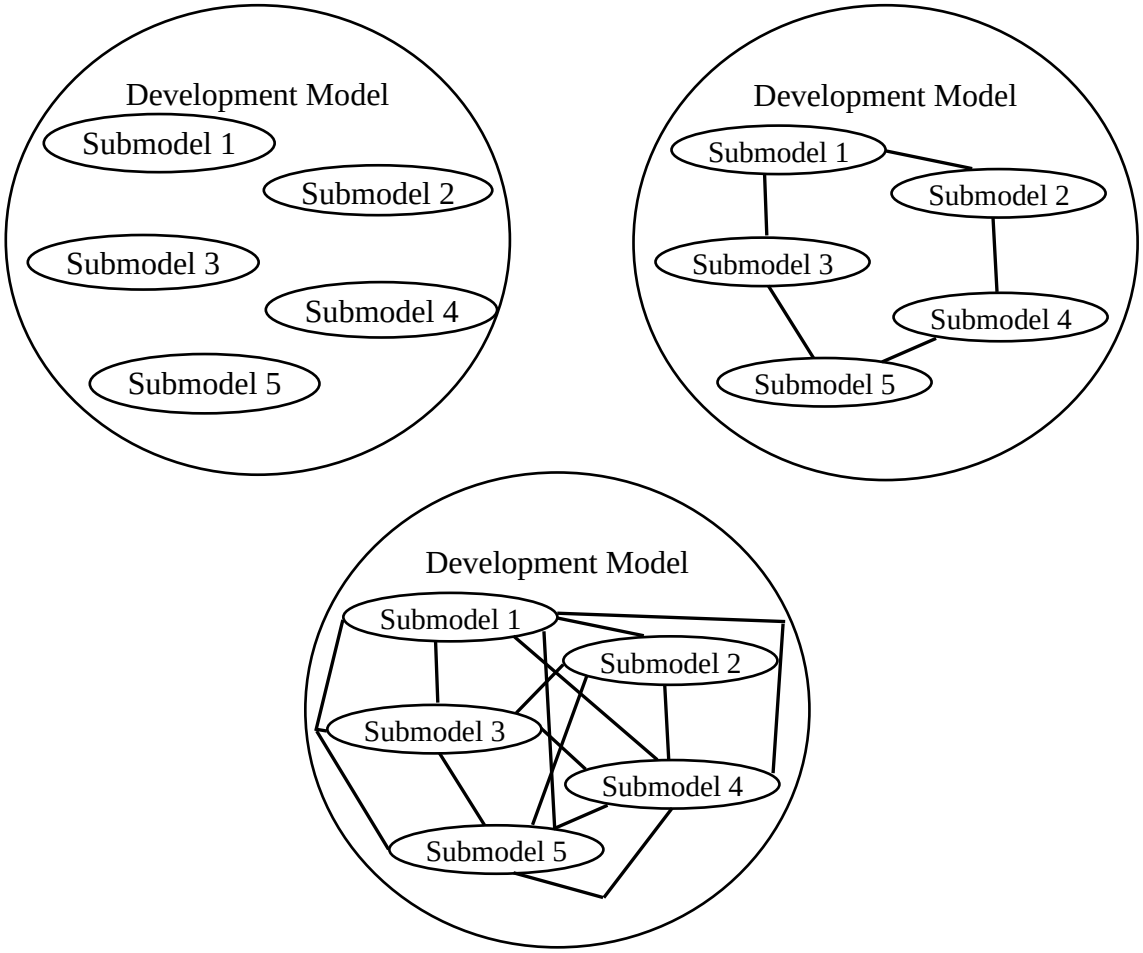

Figure 1 - Complexity description through relationships

Thus, at present the mechanism of modelling interaction among different models is not dynamic enough to be able to support MBSE to the full extent and there is a need to improve the way of modelling logic for future development in the MBSE domain. Therefore, the study tries to answer the following research questions:

- Can the interaction among the subsystems in the form of models be analysed in MBSE for solving complexity problems such as lack of communication and lack of understanding?
- Can the new dynamic ways of interaction among the subsystems improve existing static ways – hard-coded rules, pre-defined rules, relationships, and mathematical expressions?

In this paper, we propose a novel approach for answering these questions by creating a central model to govern all interactions and data exchange among different models as well as substituting pre-defined rules and relationships with a more sophisticated dynamic approach by utilising the principles of graph theory - specifically the graphical constructs known as metagraphs.

2 Enabling techniques

2.1 Graph theory

It is recognised that any type of system with various interacting components can be modelled by some kind of a graph (Basu and Blanning, 1994). Graph itself is a "representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices also called nodes or points, the links that connect some pairs of vertices are called edges also called arcs or lines" (Diestel, 2017).

The main reason behind the use of the graphs is the fact that they provide effective means to represent, visualise and maintain very complex systems that might not be possible with standard text-based or formal description methods useful only to those who can understand the corresponding formal language or familiar with the organisation of corresponding text-based document. Additionally, use of graphical structures present applicable ways to analyse structure and behaviour of complex systems.

Meanwhile, graphs are recognised as one of the best ways of model representation and utilisation. Graph visual representation is illustrated in Figure 2. For finding inconsistencies in MBSE, it is proposed that the exact matching problem of graph patterns could determine whether or not an ambiguous definition of property exists, which further extends the use of graph theory for systems modelling purposes (Herzig et al., 2014).

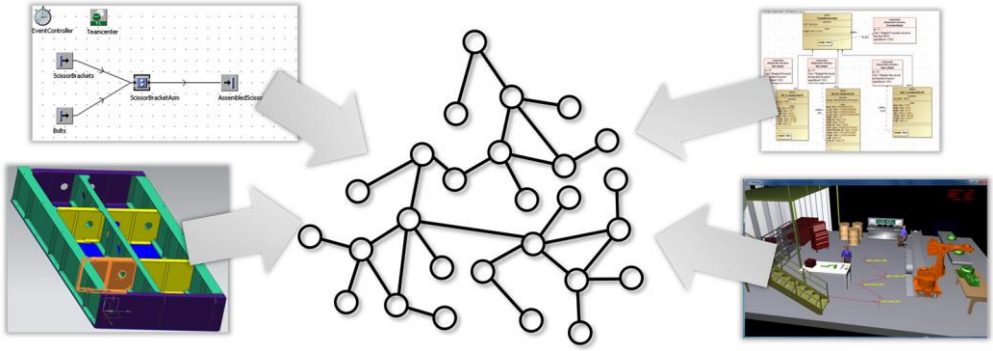

Figure 2 - Graphs as a way of model representation (Herzig et al., 2014)

There are a lot of different graphical constructs applicable to various areas. Therefore, there is a need to distinguish among them the best methodology for our purposes. To solve this problem

a comparative analysis of different graphical constructs was performed with the help of literature. For this comparison following list of criteria were identified:

- Visualisation – shows model representation capabilities.
- Directionality – implies that the current graphical construct has means for showing directions of each input-output dependencies.
- Model composition – displays whether we have enough information to determine a set of variables involved in each relationship.
- Multiple inputs/outputs – represents the capabilities of each graphical construct to deal with multiple inputs/outputs in relationships.
- Simple algebraic form – shows that graphical construct has algebraic form representation that can be utilised to some extent in programming and this graphical construct application.
- Multiple components – implies that graphical construct has developed a theory for dealing with multiple interacting components.

Based on the example set of interacting variables it is possible to perform comparative analysis of different graphical constructs that can be utilised for modelling interaction among these variables (Basu and Blanning, 1995). This example is shown in Figure 3.

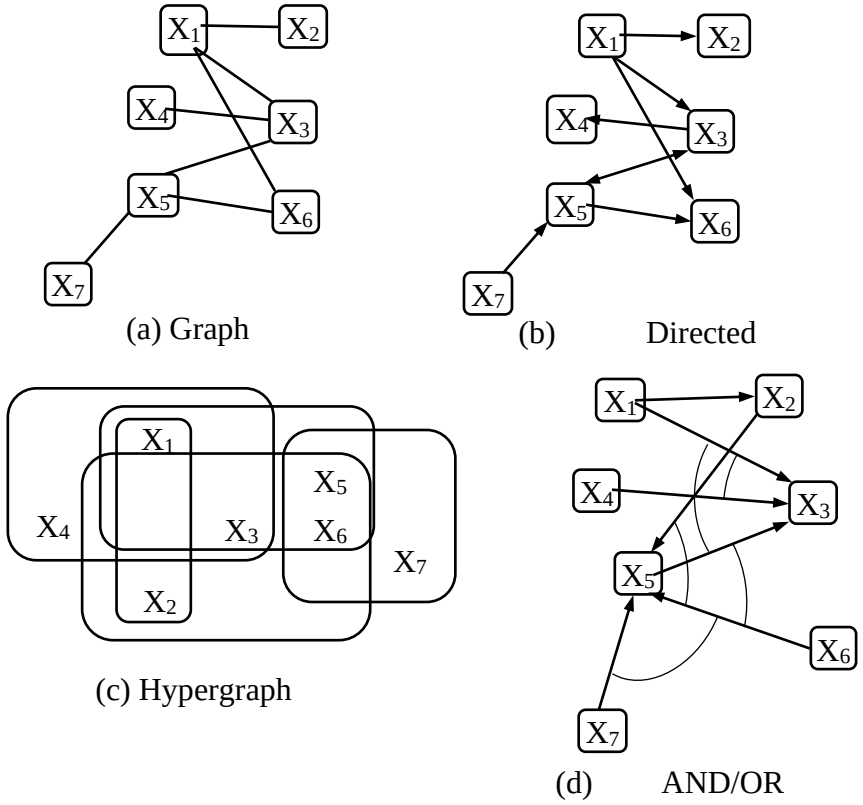

Figure 3 - Graphical representation of a set of interacting variables with (a) simple graph, (b) directed graph, (c) hypergraph and (d) AND/OR graph

Simple graph provides a good visualisation that can be utilised to show that a link between some variables exists but fails in providing information about the direction of existing relationships. Directed graph, in addition to showing the relationships among entities, includes the direction of the input-output dependencies. Thus, we know that some variables determine other variables but we still do not have enough information about the composition of models to determine particular variables.

The more sophisticated concept such a hypergraph is seen to be even more advanced way that can be utilised in systems engineering. Hypergraph handles any generalised object that represents physical or abstract properties: particles, states, points in space, etc. Analytical representation in the matrix form can be comfortably used by the computer and easily modified according to the needs of a particular system. The analysis of modern graph approaches shows that hypergraph has to be considered as one of the most adaptable tools for modelling systems as it provides additional capabilities to determine the set of variables relevant to each relationship as a single hypergraph edge covers all the variables involved in particular interaction (Gazdík, 2006). However, the hypergraph does not provide enough information to determine outputs and inputs in each relationship. A possible solution to this is to label all the variables/nodes of a hypergraph but it can be rather difficult for variables involved in different models simultaneously.

Directed hypergraphs can be used to overcome identified problems although the theory of directed hypergraphs has been developed mostly for modelling relationships among individual elements and we aim to model the interaction among a large quantity of components. AND/OR graphs combines advantages of both previous graph constructs but in some situations, where the model has multiple outputs, it requires multiple edges and becomes too difficult. Harel explains that in higraphs variables are grouped into special objects called “blobs”, which can be additionally grouped into higher-level blobs and so on (Harel, 1988). Thus, these blobs can form sets of variables or sets of sets of variables while edges connect one blob to the other. Higraph is a useful and flexible graphical structure but requires very difficult mathematical concepts to represent it.

Thus, all these constructs can be useful to some extent in numerous scientific areas but each of them fails to cover all aspects of models interaction. This includes direction representation, input/output identification and overcoming inconsistencies problems that can exist if we work with multiple models with large number of interacting variables involved in numerous models simultaneously. To solve this problem, it is possible to utilise a more sophisticated graphical construct known as metagraph.

Summary of this comparison is shown in Table 1.

Table 1 - Comparative analysis of graphical constructs

Criteria	Simple graph	Directed Graph	Hypergraph	AND/OR Graph	Metagraph	Directed hypergraph	Higraph
Visualisation	+	+	+	+	+	+	+
Directionality	–	+	–	+	+	+	+
Model composition	–	–	+	+	+	+	+
Multiple inputs/ouputs	+	+	–	–	+	+	+
Simple algebraic form	+	+	–	–	+	+	–
Multiple components	+	+	+	+	+	–	+

Plus sign in the table implies that the graphical construct has capabilities to meet the developed criteria whereas minus sign implies the opposite.

2.2 Metagraph

2.2.1 Definition

Metagraphs, an extension of directed graphs and hypergraphs, differ from conventional graphical constructs as each edge is an ordered pair of sets of elements, not an ordered pair of elements as in directed graphs or an unordered set of one or more elements as in hypergraph (Basu and Blanning, 2007). Metagraphs are considered to be a powerful method for decision support systems as they can be utilised for interaction between components analysis no matter what these components are, models, relationships or rules (Basu and Blanning, 1999).

Nodes or elements of a metagraph represent the variables and the edges represent the calculation procedure of models. Example metagraph is shown in Figure 4. This example represents the same model, which was used for comparative analysis of different graph structures.

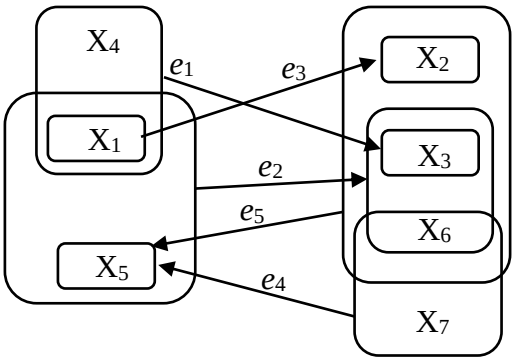

Figure 4 – Example metagraph

2.2.2 Model as a metagraph

In the metagraph view of models each model is represented as an edge with inputs as invertex and the outputs as outvertex (Basu and Blanning, 2007). Yet, the main issue is connectivity, which implies that there might be a lack of existence of one of more metapaths connecting a source set of elements to a target set of elements. Metapath is considerably different from a conventional path as edges do not have to be put in sequence, which allows the designer to model parallel calculation procedures. Also, the source and target of a metapath are sets of elements so there is no need for considering the coinput for metapaths as all the necessary information for calculation is contained inside a metapath.

This means that the corresponding models must exist if the source elements can be utilised to calculate the target elements. To overcome this issue, we need to distinguish whether there are any bridges, which are the intersection of all metapaths, and if there is more than one metapath between invertex and outvertex.

A simple example model base is illustrated in Figure 5 (Basu and Blanning, 2007). There are four variables: INFL stands for the inflation rate, REV are the revenues, EXP are the expenses and NI is the net income. Also, there are three models, which are represented by three edges: *sls* is a sales model that calculates REV out of INFL, cost model *cost* that calculates EXP from INFL and financial model *fin*, which determines NI from REV and EXP.

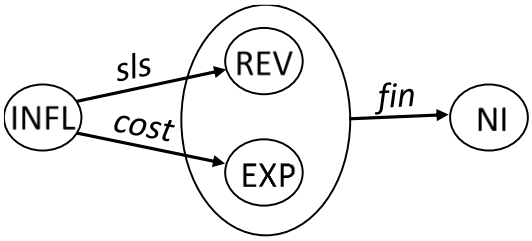

Figure 5 – Metagraph with a metapath

Table 2 shows the adjacency matrix A for this simple metagraph. From this matrix we can distinguish in a simple manner that *sls* and *cost* models do not have coinputs or cooutputs but NI column of the matrix defines that financial model *fin* takes both EXP and REV as inputs and produces NI as output, which means that EXP is a coinput of $a_{\mathrm{REV,NI}}$ and REV is a coinput of $a_{\mathrm{EXP,NI}}$.

Table 2 - Adjacency matrix A for Figure 5

	INFL	REV	EXP	NI
INFL	∅	{<∅, ∅, <sls>>}	{<∅, ∅, <cost>>}	∅
REV	∅	∅	∅	{<EXP, ∅, <fin>>}
EXP	∅	∅	∅	{<REV, ∅, <fin>>}
NI	∅	∅	∅	∅

The closure of the adjacency matrix A* is shown inм Table 3.

It completes adjacency in a way that it shows the smallest relations among the elements. From this closure, we can determine that there are only two simple paths of length more than 1 - <*sls*, *fin*> and <*cost*, *fin*> and there is no sequence of models connecting INFL to NI that is free of coinputs. For that purpose, there is a metapath {*sls*, *cost*, *fin*} connecting INFL and NI, which shows the advantage of representing model bases as metagraphs and defining metapaths that can define connectivity where simple paths fail to do so.

Table 3 - Closure A* of the adjacency matrix for Figure 5

	INFL	REV	EXP	NI
INFL	∅	{<∅, ∅, <sls>>}	{<∅, ∅, <cost>>}	{<{EXP}, {REV}, <sls, fin>>, <{REV}, {EXP}, <cost, fin>>}
REV	∅	∅	∅	{<EXP, ∅, <fin>>}
EXP	∅	∅	∅	{<REV, ∅, <fin>>}
NI	∅	∅	∅	∅

The main advantage of a metagraph is a general, non-procedural formalism that defines all properties such as reachability, connectivity and transitive closure. This formalism can be utilised to distinguish coinputs, cooutputs of any path or metapath. Also, metagraphs provide a convenient mathematical way of defining metapaths with the use of an adjacency matrix as discussed earlier.

This further shows the applicability of metagraphs for model organisation and logic management. Thus, this research currently identifies metagraphs as a basis upon which the desired framework can be most reasonably developed. Graphs extensions, such as hypergraphs and metagraphs, provide all sorts of potential to be applied for modelling systems interaction.

2.3 Design structure matrix

According to the “decomposition” principle of concurrent engineering, complex systems often can be divided into a number of more simple subsystems that are utilised to correspond to one or several tasks (Yassine and Braha, 2003). Then subsystems can be decomposed into completely independent components for further simplification of the main system model. All the subsystems and components are maintained independently, therefore modelling of individual submodels is done simultaneously. Yassine implies that proper decomposition of a complex system into simple manageable parts allows to boost product development efficiency by making engineering concurrent and easier maintainable on all phases of a life cycle.

From the decomposition principle another concept applicable to managing complexity in SE - Design Structure Matrix (DSM), which became a popular representation and dependency analysis tool. Browning identified the growing popularity of this technique (Browning, 2001). DSM represents the interactions between different components of a system in a matrix form, advantageous for logic analysis in SE. DSM itself is a square matrix, as shown in Figure 6. Rows, columns and diagonal elements are identical and stand for different system components. Each non-diagonal mark represents that dependency exists between two components in one way or another. Going across a row reveals what other elements this component affects (output sinks) while going down a column shows what other elements this component depends on (input sources).

	A	B	C	D	E	F	G	H	I
Element									
Element	●		●	●		●		●	●
Element	●	●			●	●		●	●
Element	●	●			●		●	●	●
Element	●		●	●			●	●	●
Element		●	●						
Element				●	●				
Element		●	●	●	●				
Element I	●		●		●				

Figure 6 - Example DSM

It is widely considered that the DSM approach helps to manage complexity in projects (Yassine and Braha, 2003). DSM itself is a useful matrix representation of directed graphs that can be integrated into the development process and for better system structure transparency and understanding the DSM matrix can be partitioned, which includes identification of tasks series that can be executed sequentially.

Although partitioning is an effective way for organising DSM matrix and therefore managing complexity, it is useful mostly when the elements are direct tasks. When these elements are people controlling the tasks or subsystems of a larger complex system, the process called clustering is utilised. Clusters are special combinations of elements, where all the interaction is

done inside of a cluster, minimising links to other clusters. An example of clustering is shown in Figure 7.

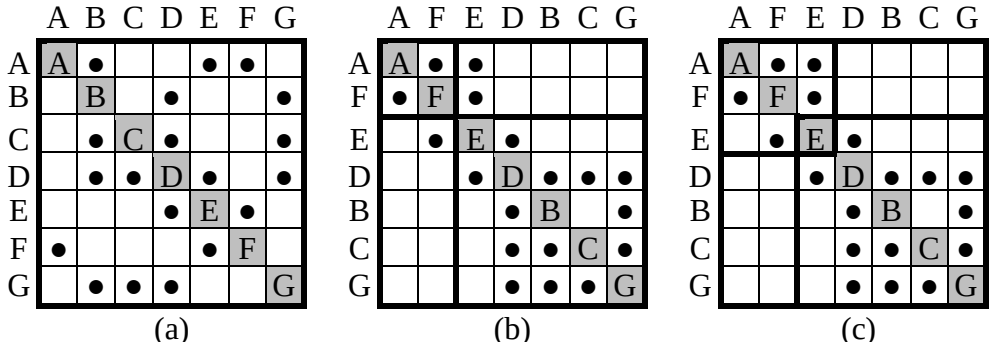

Figure 7 - Example DSM:

(a) Base DSM; (b) Clustered DSM; (c) Alternative Clustered DSM

Overall, DSM method helps to control complexity and makes a better formal representation of the interactions among different subsystems. Clustering and partitioning allow to minimise the needed quantity of iterations, which as well leads to reducing complexity. Also, DSM approach is able to help knowledge-based engineering by utilising DSM-based change propagation analysis (Tang et al., 2010). This analysis can distinguish all possible indirect dependencies or interactions among system submodels with the help of the DSM matrix. With all possible change options identified the knowledge about corresponding changes can be brought in advance, which is useful for design automation and prevents mistakes and inconsistencies in the design process. Although DSM approach is helpful for the redesign process, only structured design knowledge can be addressed in the matrix, while unstructured knowledge must be organised at first.

It is worth noting that there is a strong connection between DSM and a graph, which is shown in Figure 8. Therefore, utilising DSM in combination with metagraphs will provide further benefits for systems modelling.

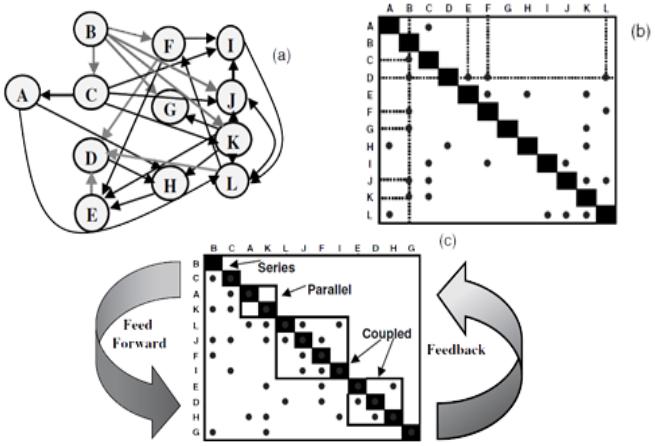

Figure 8 - Example DSM (Yassine and Braha, 2003):

Directed graph; (b) Base DSM; (c) Partitioned DSM

3 Related research

Estefan provides a survey through various existing MBSE methodologies (Estefan, 2008). Even though all methodologies are successfully utilised and provide ways of organising development process by creating models of every aspect involved, they still lack in the ways of modelling interaction among submodels and use hard-coded rules, pre-defined relationships, strict constraints, and fixed mathematical expressions. Tang identifies that in a complex product, where everything is highly dependent on something, changes in one component affects other systems and this can go further in the chain (Tang et al., 2010). Scarcity of dependency tracking methods is identified, while change propagation analysis is seen as a vital part of the whole development process. Author implies that company competitiveness depends on the efficiency of the change propagation analysis.

The other major issue in MBSE is the inconsistencies in various stages of the development lifecycle. Friedenthal recognises that design inconsistencies can arise especially when multiple people work on the same model (Friedenthal et al., 2014). To address this problem a well-defined disciplined process is proposed, which leaves space for human mistakes especially with the growing complexity of systems being developed and leads to even more inconsistencies.

Complexity in spacecraft design is identified through the discussion of graph-based design languages and the requirement of new ways for solving complex problems is distinguished (Gross and Rudolph, 2016a). In addition to that, designing modern complex systems includes solving a huge amount of problems among different interacting engineering domains (Gross and Rudolph, 2016b). Thus, it is significant to construct a proper data exchange mechanism between different engineering models to allow an easy and convenient way of propagating changes in one model to all other models and track these changes beginning in early stages with conceptual modelling. Data exchange process general view is shown in Figure 9.

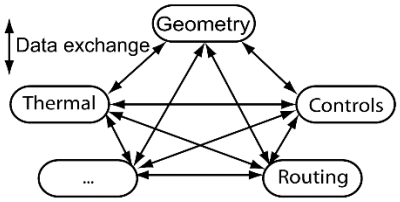

Figure 9 - Data exchange process (Gross and Rudolph, 2016b)

One way of solving this problem is the use of a central model to govern all interactions and data exchange among different models, which is shown in Figure 10.

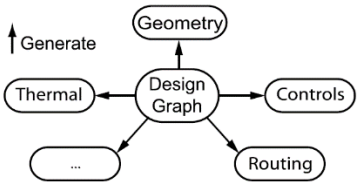

Figure 10 - Central data model (Gross and Rudolph, 2016b)

Design graph approach is introduced in application to open FireSat example and provides design graph generation for FireSat example, where nodes are instances specifications (UML Classes) and edges are links between those instances (Gross and Rudolph, 2016c). Required equations for the conceptional design of a satellite are provided by Wertz, where in graph-based design languages they are defined by edges of the design graph (Larson and Wertz, 1992). Design graph itself represents an analytical model and it is possible to analyse it in several ways, one of them being graphical representation of the solution sequence for the whole equation system. This representation provides all design graph calculated values based on input variables. FireSat example design graph is shown in Figure 11.

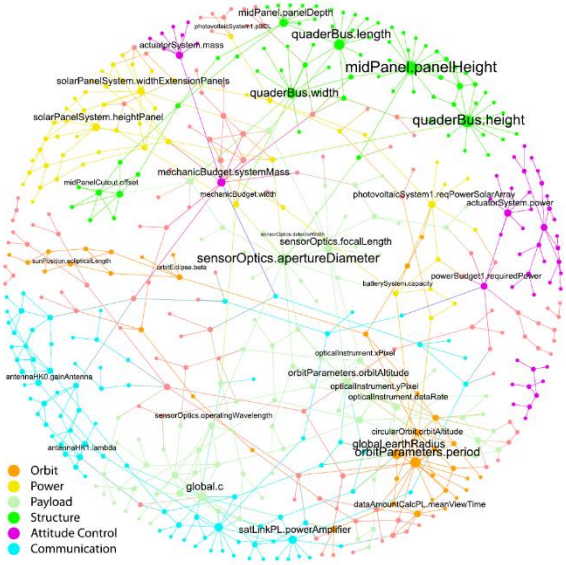

Figure 11 - FireSat example design graph (Gross and Rudolph, 2016c)

Overall graph-based design language has been demonstrated as a useful way of representing the entire design by a complex equation system revealing all the design dependencies along the design process. Keeping a large and complex system manageable by assembling all its parts by rules and dependencies is an important process. Graph view of variables involved in the development process and interactions among them provides a comfortable way of analysing systems in their current state.

Although even one missing constraint or rule can lead to the entire equation system being unsolvable with no way of searching for the error, where one minor change in one subsystem can lead to an uncontrollable chain of changes. The same is applicable to inconsistencies in the

design process, which often exists in different stages of a lifecycle. Therefore, in our study we aim to utilise such graph-based approaches that can not only represent the design in every moment but also be reliable and help design engineers to keep track of all the changes on every stage of a lifecycle automatically.

A number of features are needed in representing interactions among multiple models such as directionality representation, visualisation capabilities, understanding sets of variables involved in each relationship and formalised algebraic representation (Basu and Blanning, 2007). Thus, metagraphs provide all these features and thereby prove to be appropriate for the intended study.

It was also identified from the literature that metagraphs can be applied to data and rule management, where rule bases can be represented as metagraphs and integrated into models. Moreover, workflows and processes can be modelled with the use of metagraphs. Metagraphs are considered to be a powerful method for decision support systems as they can be utilised for interaction between components analysis no matter what these components are, models, relationships or rules (Basu and Blanning, 1999).

4 Methodology design

4.1 Requirements

The requirements for the methodology being developed are subdivided into two categories as functional and design. Functional requirements are responsible for the correct practical implementation of the developed methodology. These requirements are as follows:

- Effectiveness – this requirement states that the developed method will allow effective modelling of systems interaction in MBSE environment. Effectiveness is defined and measured against a certain set of metrics during the verification and validation stage.
- Automatic control over interactions – this is important so that modelling logic in the form of subsystems interaction is dynamic and eliminates inconsistencies of multiple definitions of the same relationships according to certain rules or automatic hints and helps to diminish the time needed for the development of products.
- Systems interaction representation – requirement related to correct and simple representation of needed relationships from different points of view.

Design requirements provides the methodology from the developer’s point of view and its other users. These requirements are as follows:

- Special knowledge – there is a need for a special systems interaction engineer who will have knowledge of the composition of the interaction module and will be able to control this module in case there is a need for it.
- Relationships models – it is necessary to provide systems interaction engineer full capabilities and a range of building blocks/models to be able to control automatically built interaction module.
- Applicability for multiple users – this requirement states that different users should be able to utilise the developed methodology. These users involve engineers of different expertise involved in various stages of the development process. Not all of these engineers have adequate knowledge on how the actual system's interaction is modelled inside the interaction module of the main system model but still, they have to be able to define relationships between their work and other engineers.
- Friendly user experience – this requirement related to both users and the systems interaction engineer as they should be able to seamlessly utilise developed framework with the help of software with friendly and simple user interface. Also, the framework should provide simple ways of representing certain relationships from different points of view, which is set to be achieved with the help of special graphical representation software.

All design requirements are related to certain functional requirements as the developed method is set to be fully capable of simultaneously modelling interactions among models and providing means and tools to control and utilise the interaction model by all its users.

4.2 The interaction modelling framework

A framework for developing interaction mechanism, as shown in Figure 12, consists of five phases, namely

1. System definition
2. System Modelling
3. Interaction Modelling
4. Validation and verification
5. Visualisation.

The framework provides a set of activities for MBSE experts to employ in order to implement graph-based systems interaction technique in the development process.

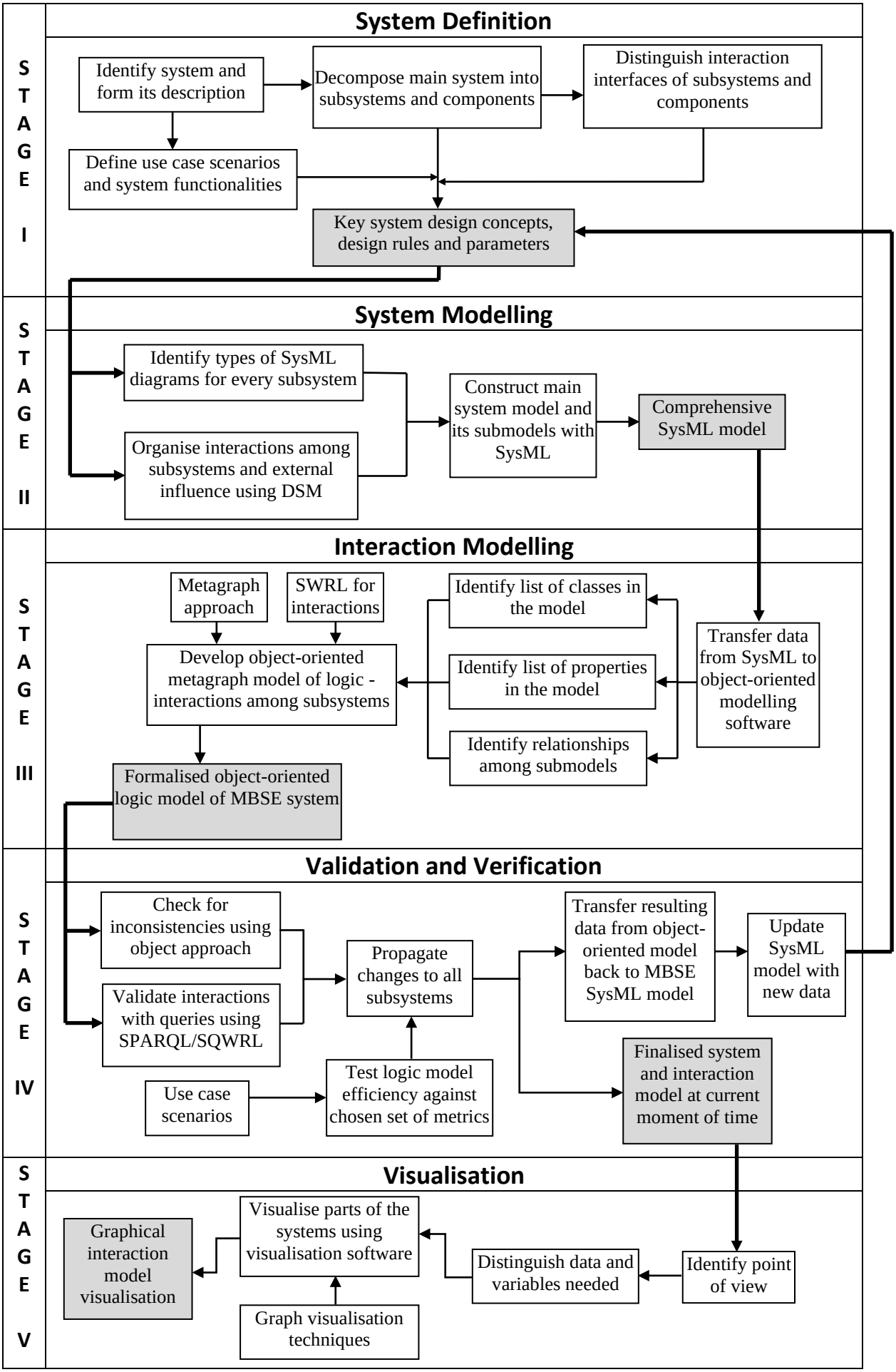

Figure 12 - The framework for interaction modelling

The most significant part of the framework is metagraph construction, which automatically governs the actual interactions among different components.

The first phase is to define the system - this includes decomposing main system into subsystems and components while defining corresponding interaction interfaces and system functionalities. It is essential for correct system modelling to adequately distinguish key design concepts, rules, and parameters.

The second phase is the system modelling itself, where all subsystems identified in the previous are modelled with the use of SysML diagrams resulting in the creation of a comprehensive SysML model.

The third phase is where interaction modelling happens. Data is set to be transferred from SysML to object-oriented modelling software, where metagraph approach is utilised to successfully model the interaction mechanism in the system being developed. The data from SysML is exported into XML-file and parsed using the object-oriented approach. XML has been widely adopted as a universal way to format data. LINQ to XML parsing technique is utilised to analyse and parse XML data. This technique is a fast, forward-only, non-caching parser with a lot of useful built-in programming functions and guides. Therefore, it allows the user to quickly find and parse needed information from the XML file. Also, it provides a good way to construct and populate new XML trees. Object-oriented approach grants capabilities to enhance knowledge categorisation and to model relationships among systems in a simple object-oriented graph-based manner. Object-oriented modelling software such as Microsoft Visual Studio provides a good programming interface and includes ways to validate model consistency through reasoning.

The fourth phase is the validation and verification phase, where the developed model is checked for consistency and correct representation of the initial model. If something changes in one of the subsystems of the initial model and its SysML representation the object-oriented metagraph interaction model checks the updates for consistencies and propagates changes to other subsystems ultimately transferring data back to SysML and updating corresponding diagrams.

The final visualisation phase is to provide a method to represent the model at any moment from different points of view with the help of graph visualisation software.

5 Use cases

The methodology framework is designed to be tested in a series of use cases. The first use case is the Automobile Example from literature (Friedenthal et al., 2014). In this use case, the model-based systems approach is applied to design an automobile system, where the final is set to

match the acceleration and fuel efficiency requirements. The full range of Automobile design systems is modelled in SysML and the full list of models is shown in Table 4.

Table 4 - Full list of models in the Automobile Example

Diagram name/System	**Diagram Type**
Model Organisation	Package diagram
Automobile System Requirements	Requirement diagram
Automobile Domain	Block definition diagram
Operate Vehicle	Use case diagram
Drive Vehicle	Sequence diagram
Turn On Vehicle	Sequence diagram
Control Power	Activity diagram
Drive Vehicle States	State machine diagram
Vehicle Context	Internal block diagram
Vehicle Hierarchy	Block definition diagram
Provide Power	Activity diagram
Power Subsystem	Internal block diagram
Analysis Context	Block definition diagram
Vehicle Acceleration Analysis	Parametric diagram
Vehicle Performance Timeline	Timing diagram (not SysML)
Engine Specification	Block definition diagram
Max Acceleration Requirement Traceability	Requirement diagram
Architect and Regulator Viewpoints	Package diagram

Relevant list of model components has been distinguished from the full list of models. This example was modelled with the use of Visual Paradigm free modelling software for SysML modelling.

The metagraph model of this SysML example has been derived in theory and shown in Figure 13.

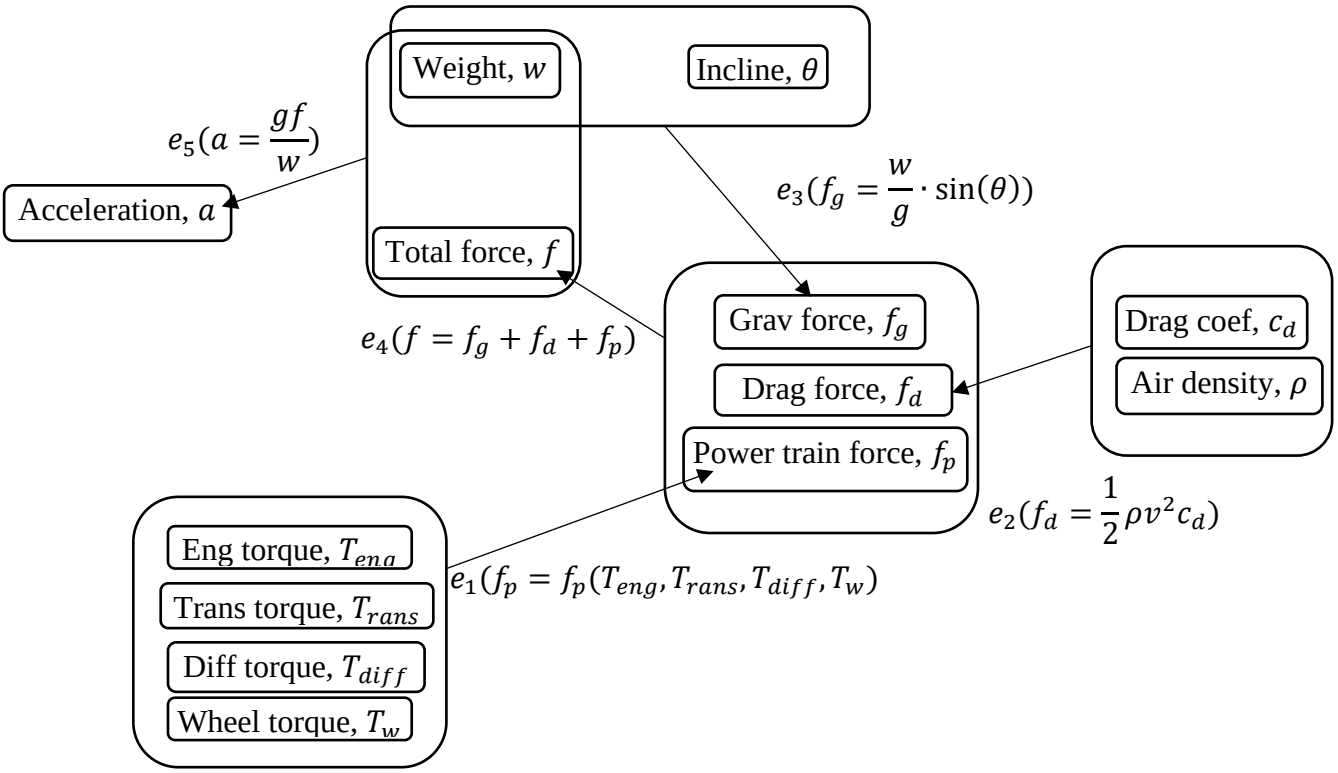

Figure 13 - Metagraph for the Automobile Acceleration Example

Then the SysML model is exported in the XML format to a file that is then parsed using object-oriented approach. For that purpose, the LINQ to XML open library in C# is utilised. This code is also used to automatically generate the Metagraph model from the source XML file. Currently, the program doing this automatic metagraph generation has already been developed with the user interface for testing purposes.

Several scenarios for testing the proposed framework have been identified. These scenarios include the following cases: changes of specific parameters (changing values), changing in one of the models (changing relationships between different components) and changing of one of the models (swapping equations with some other dependencies). Now the process of testing these scenarios is ongoing using the self-written C# code mentioned earlier.

Graphviz has been identified as the visualisation technique as it has all the needed capabilities starting from automatic import from C# and automatic visualisation of the exported graph structure from the developed code.

The second use case is supposed to be the FireSat example from literature. The FireSat mission describes a satellite flying in low earth orbit for earth observation in the infrared band. The payload of the satellite is supposed to observe forest fires in the US. The model of the satellite is supposed to enable the layout of most of the subsystems on a conceptual level. In the following sections, a selection of some class diagrams from the model are presented along with short descriptions of their purpose (Groß and Rudolph, 2012).

6 Discussion

As stated in section 1, the aim of the research is to improve existing methodologies and interaction modelling as well as deal with inconsistencies of the design process. Methodology framework is the expected contribution to the knowledge of the method being developed. With systems and models becoming more and more complex in systems engineering domain, issues of overcomplexity, lack of understanding and communication problems in MBSE lead to many challenges in the design process. Developing a new way of modelling logic in MBSE and improving over existing ways tackle all kinds of these problems and push forward the state of systems engineering field. The presented framework for graph-based modelling of systems interaction in MBSE environment is set to help systems engineers to develop better quality less expensive products in less time. This work may be viewed as a step forward toward more consistent and automatic modelling of interactions among subsystems and components in MBSE. This also involves making a way of communication, which dynamic in nature by providing more automatic ways of interaction.

In addition to the identified advantages for SE field, the research being undertaken thus far is seen to confidently identify potential research proposals for future work. The proposed framework is set to be tested against a chosen set of metrics within different use cases. The methodology provides a common and consistent basis for future interaction modelling among subsystems in various aspects of MBSE field.

These principles have potential utilisation for design automation purposes in model-based systems engineering field. Automation techniques always have multiple applications in systems engineering field and specifically MBSE as engineers always aim to produce higher quality and less expensive products in less time achieving this by integrating knowledge on every stage of a lifecycle. Potential implementation of developed dynamic logic modelling methods to MBSE through is set to provide improvements for engineering methodologies.

7 Conclusion

A range of MBSE methodologies are considered useful in developing various complex products and helping system engineers maintain systems complexity on a different stage of the development lifecycle. It is supported by the comparative analysis of MBSE methodologies done by Estefan (Estefan, 2008). Even though all of them are successfully utilised in MBSE field to some extent, the systems interaction is still mostly done manually with hard-coded rules, pre-defined relationships and constraints (Tang et al., 2010).

In addressing the above issues, this paper has proposed a novel framework that aims to improve the interaction mechanism in model-based systems engineering by developing new dynamic

ways of modelling logic. To model the interactions the authors propose to use graph-based approach with metagraphs and object-oriented modelling being in its core. Metagraph theory is a powerful tool already being utilised in decision support systems (Basu and Blanning, 1999). We believe that the framework is set to be applicable to various systems being developed and may be considered as a step forward in the development of systems engineering field. This approach will provide improvements for design automation and MBSE field as one of the most advanced approaches nowadays. The initial methodology framework idea has been presented during the SECESA 2016 and published in the conference proceedings (Filimonov et al., 2016).

Future work involves evaluating the framework through use cases. These use cases are various real-life development processes such as the development of a car or its systems and can be obtained from the open literature sources. Based on the initial modelling results, it has already been identified that the proposed methodology framework complies with the distinguished requirements from Section 4.1. The required adjustments will be made, and the framework will be evaluated in the more complicated use cases as the next step of this research.

References

Bahill, T., Botta, R., 2008. Fundamental Principles of Good System Design. Eng. Manag. J. 20, 9–17.

Basu, A., Blanning, R.W., 1994. Metagraphs: A tool for modeling decision support systems. Manage. Sci. 40, 1579–1600.

Basu, A., Blanning, R.W., 1995. Metagraphs. Omega 23, 13–25.

Basu, A., Blanning, R.W., 1999. Metagraphs in workflow support systems. Decis. Support Syst. 25, 199–208.

Basu, A., Blanning, R.W., 2007. Metagraphs and their applications. Springer Science & Business Media.

Browning, T.R., 2001. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. Eng. Manag. IEEE Trans. 48, 292–306.

Diestel, R., 2017. Graph theory. Springer Publishing Company, Incorporated.

Estefan, J.A., 2008. Survey of Model-Based Systems Engineering (MBSE) Methodologies.

Filimonov, M., Raju, P., Chapman, C.B., 2016. Graph-based modelling of systems interaction in model-based systems engineering environment. In: 7th International Systems & Concurrent Engineering for Space Applications Conference Proceedings. Madrid.

Friedenthal, S., Moore, A., Steiner, R., 2014. A practical guide to SysML: the systems modeling language. Morgan Kaufmann.

Gazdík, I., 2006. Modelling systems by hypergraphs. Kybernetes 35, 1369–1381.

Gross, J., Rudolph, S., 2016a. Modeling graph-based satellite design languages. Aerosp. Sci. Technol. 49, 63–72.

Gross, J., Rudolph, S., 2016b. Geometry and simulation modeling in design languages. Aerosp. Sci. Technol. 54, 183–191.

Gross, J., Rudolph, S., 2016c. Rule-based spacecraft design space exploration and sensitivity analysis. Aerosp. Sci. Technol. 59, 162–171.

Groß, J., Rudolph, S., 2012. Generating simulation models from UML-A FireSat example. In: Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium. p. 25.

Harel, D., 1988. On visual formalisms. Commun. ACM 31, 514–530.

Haskins, C., 2006. INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, v. 3. John Wiley & Sons.

Herzig, S.J.I., Qamar, A., Paredis, C.J.J., 2014. An approach to identifying inconsistencies in model-based systems engineering. Procedia Comput. Sci. 28, 354–362.

Holt, J., Perry, S., 2008. SysML for systems engineering. IET.

Larson, W.J., Wertz, J.R., 1992. Space mission analysis and design.

Shekar, B., Venkataram, R., Satish, B.M., 2011. Managing complexity in aircraft design using design structure matrix. Concurr. Eng. 19, 283–294.

Systems Engineering Vision 2020, 2007. . INCOSE-TP-2004-004-02, Version 2.03.

Tang, D., Zhu, R., Tang, J., Xu, R., He, R., 2010. Product design knowledge management based on design structure matrix. Adv. Eng. Informatics 24, 159–166.

Wymore, A.W., 1993. Model-based systems engineering. CRC Press.

Yassine, A., Braha, D., 2003. Complex concurrent engineering and the design structure matrix method. Concurr. Eng. 11, 165–176.