SysML Model Transformations Using Relational Orientation

C.E. Dickerson, S. Ji, M.K. Wilkinson

Mathematical Formalism DSIG

12th of September 2022

GENERAL USE RESTRICTIONS

REFERENCES

[1] C. E. Dickerson and S. Ji, *Essential Architecture and Principles of Systems Engineering*, Boca Ratton: CRC Press, 2021.

[2] OMG, *MDA Guide v2.0*, 2014.

[3] OMG, *UPR: UML Profile for ROSETTA*, v1.0, 2019

[4] C. E. Dickerson *et al*., "Architecture Definition in Complex System Design Using Model Theory," in *IEEE Systems Journal*, vol. 15, no. 2, pp. 1847-1860, June 2021.

[5] OMG, *Systems Modeling Language (SysML™)*, v1.6, 2019.

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

[7] L. Delligatti, *SysML distilled: A brief guide to the systems modeling language*. Addison-Wesley, 2013.

[8] M. Wilkinson and T. Rabbets, *Don't Panic - The Absolute Beginner's Guide to Architecture and Architecting*, INCOSE UK, Ilminster, 2020.

[9] M. Wilkinson, *A Systems Journey – From Theory to Practice and Back Again, Keynote Address*, INCOSE Conference on Systems Engineering Research (CSER), 2022.

[10] *Systems and software engineering – System life cycle processes*, ISO/IEC/IEEE 15288:2015, 2015.

[11] *Systems and software engineering – Architecture description*, ISO/IEC/IEEE 42010:2011, 2011.

[12] C. E. Dickerson, S. Ji and R. Roslan, "Formal methods for a system of systems analysis framework applied to traffic management," *2016 11th System of Systems Engineering Conference (SoSE)*, 2016, pp. 1-6.

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

The purpose of this paper is to offer a distilled summary of transformations of graphical models that have been expressed in SysML diagrams, from Use Cases through Sequence Diagrams. Model transformations using a mathematically based relational orientation method have been taken from [1] and are summarized in Section 2. The figures are under copyright but are used with permission. All figure numbers in this paper are taken from the original source for clarity.

In its mission to develop mathematically based model transformations for OMG standards, the Mathematical Formalism DSIG has drawn from concepts in OMG standards such as Model Driven Architecture [2] complemented by an extensive body of non-normative references in Section 3.2 of UPR 1.0 [3], recently published research [4], and SysML related publications such as [5-7].

1.2 Context

The model transformations presented in this paper reflect the achievements of more than a decade of research with industrial end users of MBSE including but not limited to Jaguar Land Rover, Rolls-Royce, and BAE Systems. These transformations mark the practical completion of the 2nd objective for system architects in the Math DSIG mission statement [https://www.omg.org/maths/].

The next objective of the DSIG focuses on expressing the underlying mathematical formalisms for MBSE via OMG model-based standards. Details are proposed in Section 3. With previous success in the standardization of relational transform through the adoption of UPR 1.0, it is envisioned that the goals of the OMG Mathematical Formalism DSIG are achievable. However, to ensure success, we believe that engagement with authoritative communities beyond the OMG is essential. These communities include for example, the International Council on Systems Engineering (INCOSE) and the International Organization for Standardization (ISO).

A link has already been established with INCOSE via the Architecture Working Group (AWG) of the INCOSE UK Chapter, whose programme of work aims to clarify and formalize architecture concepts and improve architecting methods. In addition to contributing to international standards, the group and its members have published a variety of practitioners guides and an introductory book [8]. Current activities include workstreams focused on formalization through modelling, and this work is being supported by the authors of this paper, who are contributing their foundational research [4] to the endeavor. A contextual summary of the work of the AWG was provided in a keynote address at the CSER 2022 conference [9].

1.3 Definition of Key Terms

In 2020 [4] the authors of this paper published the results of several years of research and engagement with the ISO. The purpose of the 2020 paper was to first offer definitions of key terms used in the current ISO standards for systems and software engineering [10, 11] that had a consistent mathematical basis. These were necessary for the mathematical specification of an architecture definition technical process.

Building on [4], Dickerson and Ji [1] offered further analysis of the terms and comparative discussions with alternative definitions. These details and be found in [1] and [4] for the terms *Structure, Architecture, System, and Model.* For the term *Semantic Structure*, the concept had been introduced as early as 2013 but not elaborated in detail until 2016 in an IEEE conference paper [12].

The reader is cautioned that the use of language in these terms is mathematically oriented. The usage is similar to software engineering but for this reason there are subtle differences with the object-oriented usage of fundamental terms such as class, object and relation. With this in mind the following four definitions are offered without further motivation or explanation.

> *Structure* is junction and separation of the objects of a collection defined by a property of the collection or its objects.
>
> *Architecture* is structural type in conjunction with consistent properties that can be implemented in a class of structure of that type.
>
> In first order model theory, a *Model* is a relational structure for which the interpretation of a sentence in the Predicate Calculus becomes valid (true).
>
> A *System* is a set of interrelated elements that comprise a whole, together with an environment.
>
> *Semantic Structure* is a structure whose objects are semantic types*.*

Each of the semantic structures used in Section 2 will be the 'clean' structure of an object-oriented diagram in UML/SysML such as a Use Case Diagram, Activity Diagram, or a Sequence Diagram. However, for the sake of simplicity and clarity, domain knowledge from a tutorial on an air traffic control system [1] will be interpreted into the semantic structures to create domain structures. The underlying model definition and transformation process employed in Section 2 does not depend on the knowledge details of the domain structures.

1.4 Relational Orientation

Overview. Relational orientation is an architectural technique for abstraction that expresses the concepts of a system in terms of relations among elements. In first order model theory, a relational structure is a collection of mathematical relations (in the sense of set theory). Concepts are expressed formally using the (first order) predicate calculus. A first order model is defined to be a relational structure that is the image of an injective mapping of one or more sentences in the first order predicate calculus (a sentence being a fully quantified well-formed formula). In other words, the model 'faithfully' realizes the sentences. Relational orientation is then a technique that supports the modelling of systems.

The relational orientation on systems supports a general systems methodology that employs a principle of model specification and relational transformation for the purpose of system description, analysis, and design. In relational orientation, the specification of a model associated with a system is the specification of:

- Entities associated with the system
- Sentences (declarations) about the entities
- Modeling elements to instantiate the sentences
- A semantic structure on the modeling elements
- Interpretations of the sentences into the semantic structure

Entities are abstractions that admit logical or physical existence. The entities of the system can include elements, classes (of elements), and properties of the system e.g., functionality. Note that there may be entities associated with the system which are not part of it, e.g., elements of the operating environment and context of the system. The sentences are the basis for system specification. A system model is valid when the interpretation of each sentence it represents is true within the semantic structure of the model.

Refer to UPR 1.0 [3] Section 6.2.1 and the references there for further details and explanation.

Application to object-oriented graphical models. A graphical representation of a model is a collection of vertices and edges for encoding the semantic information captured by the sentences. The modeling elements in this case are the vertices. The edges, which represent relationships between vertices, are represented as pairs of vertices. The underlying structure is a (discrete) graph and therefore also has a matrix representation. It is clear then that the object-oriented graphical models of UML and SysML can be represented by matrices.

A simple example is in the figure below. The Design Objectives y_1, y_2, and y_3 could be three activities from an Activity Diagram in which the first two must be performed before the third one. This defines two binary relations which are represented as two ordered pairs (y_1, y_3) and (y_2, y_3) based on the usual (row, column) notation for cells in the matrix (M). If the entities of the Target Model x_1, x_2, and x_3 are three system elements; and the transformation matrix (Q) is the functional allocation of y_1 to x_1 , y_2 to x_2 , and y_3 to x_3 then the relations in the matrix (M) are transformed into the matrix (N).

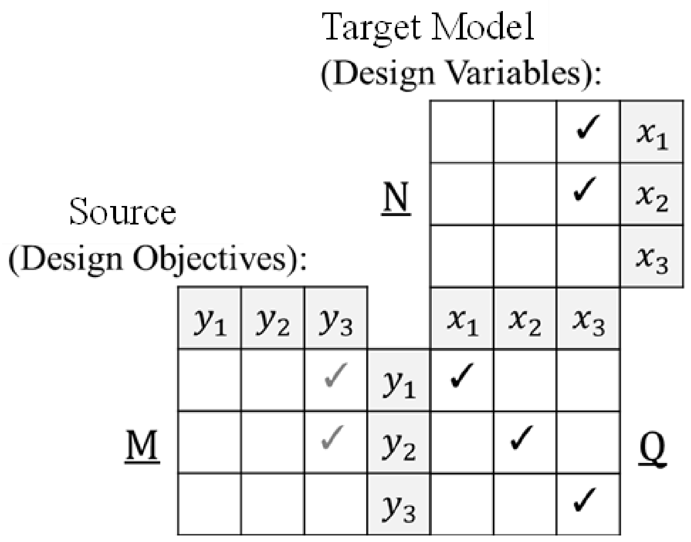


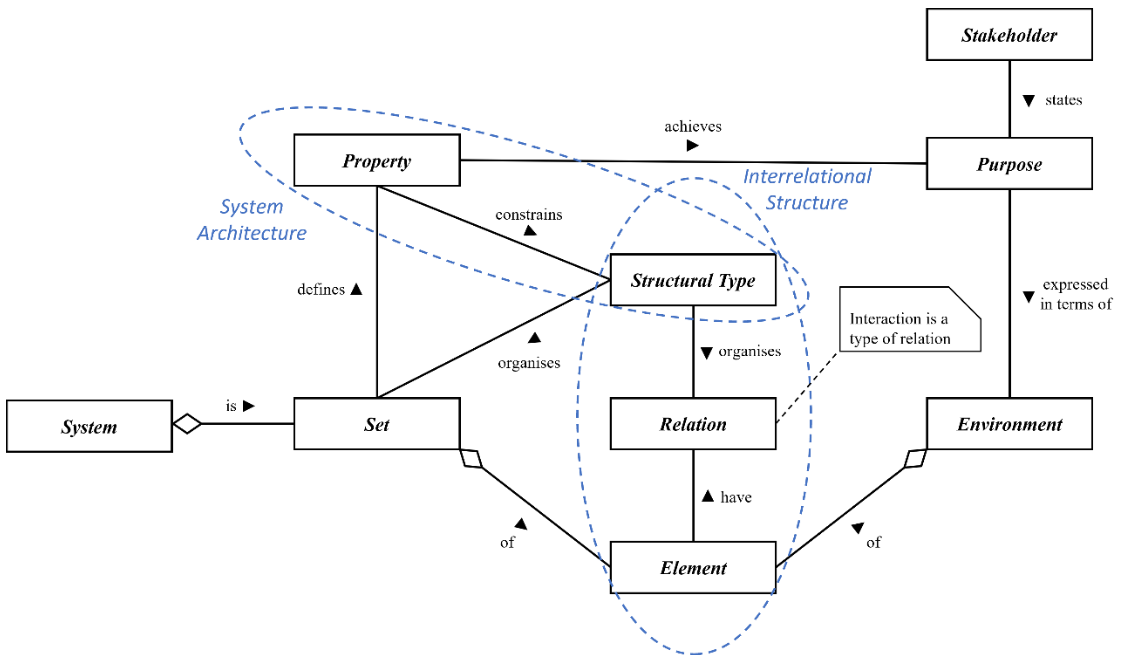
Figure 6-1 Simple example of a Transformational Frame (Q) in matrix form [3]

The transformation is computed according to the formula,

$$(y_i, y_j) \in \boldsymbol{M} \text{ with } (y_i, x_k), (y_j, x_l) \in \boldsymbol{Q} \text{ implies } (x_k, x_l) \in \boldsymbol{N}$$

The Relational Oriented Systems Engineering Technology Tradeoff and Analysis (ROSETTA) framework is a matrix representation of the relational orientation on systems. The figure above is a simple example of how it can be applied to the object-oriented graphical models of UML and SysML. ROSETTA provides a facility for interpretation of concepts between system architects and systems (and software) engineers, among other things. The name of the (mathematical) framework bears an intentional similarity to the Rosetta stone which provided the means to interpret between the Greek, Egyptian and Hieroglyphics demotic languages.

Integrated model of architecture and system. Figure 4.11 from [1] provides a logical model of an integration of the definitions in Section 1.3 for architecture and system. More precisely, it is a semantic structure that can be populated with domain knowledge. The graphical nature of the structure exposes the binary predicates (relations) of the terms in the definitions of architecture and system. Because the interpretation of (predicate) sentences into relational structures is a first order model, the semantic structure in figure 4.11 becomes the basis for defining models of a system of interest and the relational transformations between them. This will be exploited extensively in the next section.



©Figure 4.11 Integrated model of architecture and system [1]

2. Model Types and Transformations

Object-oriented diagrams in SysML provide a collection of model types for defining the models of a system of interest and transformations between them. The 'clean' structure (i.e., without domain knowledge) of an object-oriented diagram in UML/SysML expresses a model type. Taking a relational orientation, ROSETTA provides a method for a model definition and transformation process that does not depend on the knowledge details interpreted into the semantic structures. However, for the sake of simplicity and clarity, domain knowledge from a tutorial on an air traffic control system [1] will be interpreted into the semantic structures to create domain structures.

The object-oriented model types used in this section are based on: the Use Case Diagram, the Activity Diagram, Class or Block Diagrams, and the Sequence Diagram. There is an abundance of well-known literature with examples and guidance for the drawing of diagrams. Friedenthal [6] and Delligatti [7] are two popular examples. The methods of these two books generally fall into the category of object-oriented systems engineering.

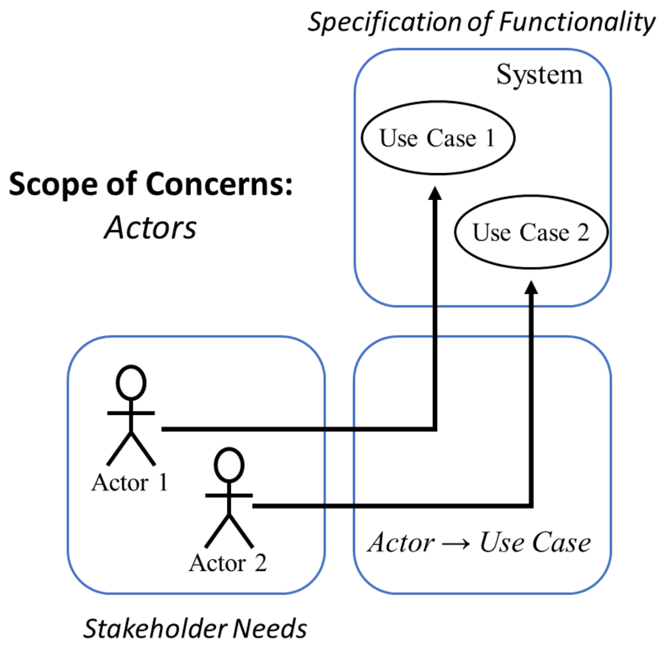
The methods for transforming between the diagrams as first order models is one of the primary concerns of ROSETTA [3] and is demonstrated in detail in the tutorial chapters of the book by Dickerson and Ji [1]. The methods in [1] and [3] fall into the category of relational-oriented systems engineering. The feasibility and practicality of the relational approach has been demonstrated in multiple deliveries of post graduate systems engineering modules as well as government sponsored research and commercialization projects. Refer to Annex A-4 of [1] for a distilled summary.

Comparison of the object and relational oriented approaches is beyond the scope of this paper.

In the remainder of the section, the sequence of ordered parings of models and transformations between the models is a realization of the mathematically defined Architecture Definition technical process in [4]. It follows a 'top-down' approach for the sake of logical flow but in practice transitions between two ordered pairings of models can be sequenced in whatever manner is practical.

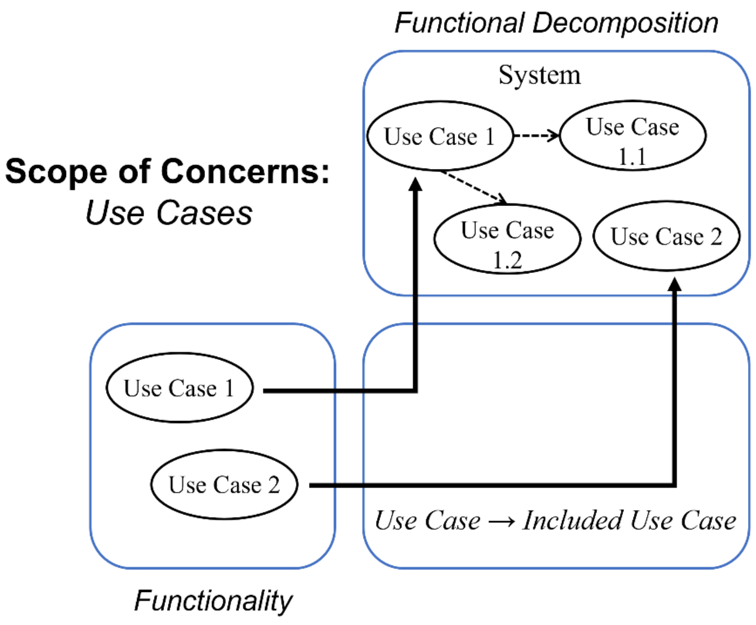
2.1 Functionality to Behavior

This subsection shows how to transform object-oriented models beginning with system functionality represented by Use Case Diagrams and concluding with the specification of use case description tables that map to Activity Diagrams.



©Figure 5.5 Specification of system functionality [1]

The representation of system functionality by a Use Case Diagram is actually a transformation of a model of the actors in the system environment into properties of the system. Figure 5.5 [1] illustrates a transformation of an external view of the system boundary into use cases. Associations between actors in the environment and (functional) properties of the system (as a black box) are based on interactions.



©Figure 5.6 First-level functional decomposition [1]

Functional decomposition can be accomplished by specifying included use cases. The source model in Figure 5.6 [1] is taken exactly as specified in the Use Case Diagram. If the functionality represented by a use case is decomposed into lower level functionalities, then the inclusion relation can represent the decomposition. Thus, in the figure,

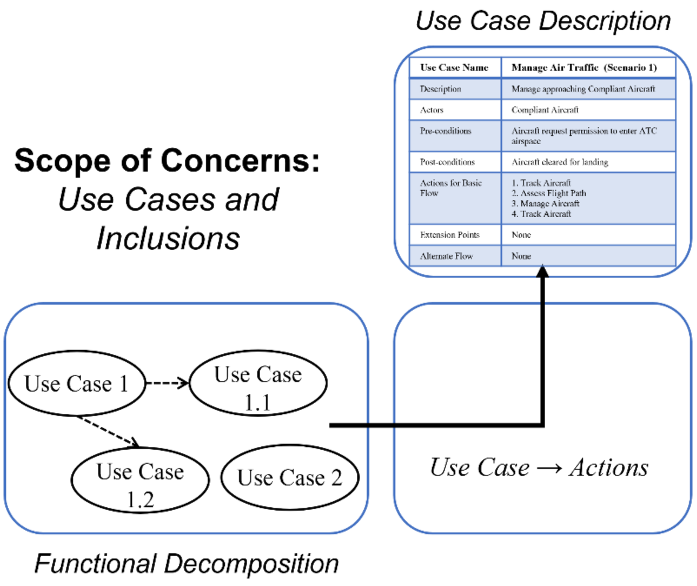
Use Case 1 → [(Use Case 1, Use Case 1.1); (Use Case 1, Use Case 1.2)]

Each inclusion relation creates an ordered pair but there is no implication of ordering between the pairs of inclusions.

Also in the figure, there is no decomposition of the functionality represented in Use Case 2. The semantic transformation then acts on this use case as an identity mapping:

Use Case 2 → Use Case 2

Note that in these first two examples of model specification and transformation, the figures that represent the process are for the semantic structures themselves with no reference to domain knowledge. In the examples in the remainder of Section 2, domain knowledge will be added in order to help with visualization of the process.



Use Case Name	Manage Air Traffic (Scenario 1)
Description	Manage approaching Compliant Aircraft
Actors	Compliant Aircraft
Pre-conditions	Aircraft request permission to enter ATC airspace
Post-conditions	Aircraft cleared for landing
Actions for Basic Flow	1. Track Aircraft 2. Assess Flight Path 3. Manage Aircraft 4. Track Aircraft
Extension Points	None
Alternate Flow	None

©Figure 5.7 Specification of Use Case description [1]

When the functionalities in the use cases are decomposed to a level that is actionable, they can be associated with actions that form the basic flow of actions through the system. This is the basis for an Activity Diagram. Rather than transform from the functional decomposition, it will be useful to make a direct mapping of the included use cases (when they are actionable) into a structured table that is commonly known as a Use Case Description. This table is a sematic structure.

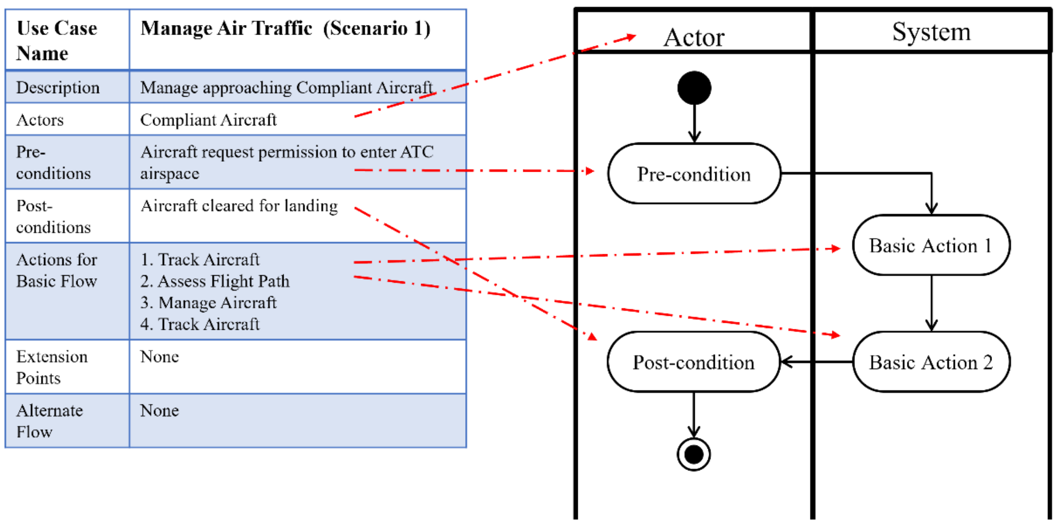
Figure 5.7 [1] depicts such a mapping for the Air Traffic Control System (ATCS) that is the subject of the tutorial chapters of [1]. When domain knowledge is interpreted into Use Case 1, it becomes 'Manage Air Traffic'. There are three included use cases in the tutorial and they each are actionable:

Use Case 1.1: Track Aircraft

Use Case 1.2: Assess Flight Path

Use Case 1.3: Manage Aircraft

The source model in Figure 5.7 [1] can be revised to reflect the three actionable included use cases. These are then mapped to the three actions that form the basic flow of actions through the system. The table represents the processes of the ATCS in a baseline scenario in which the system would operate under intended conditions.



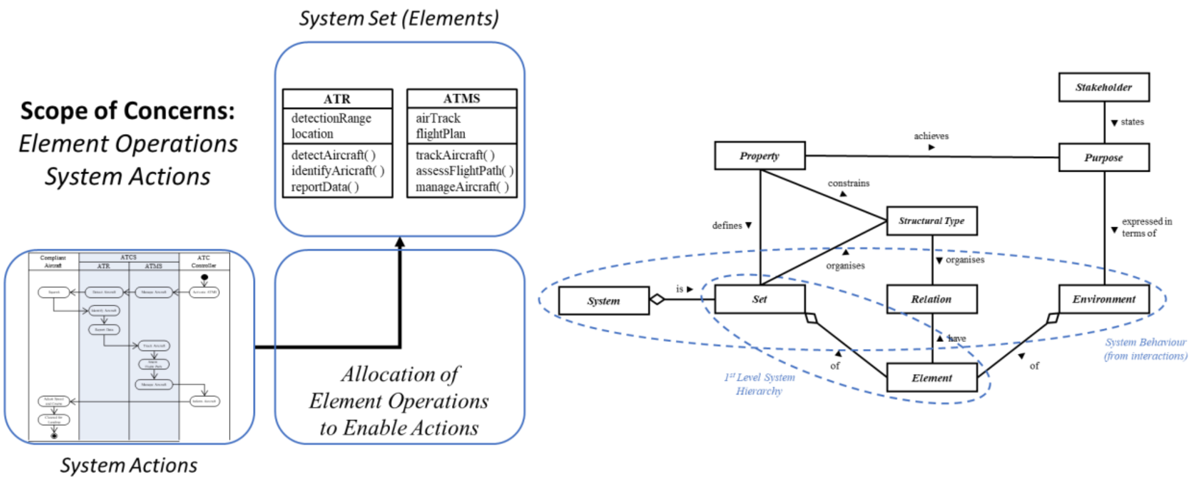
Use Case Name	Manage Air Traffic (Scenario 1)
Description	Manage approaching Compliant Aircraft
Actors	Compliant Aircraft
Pre-conditions	Aircraft request permission to enter ATC airspace
Post-conditions	Aircraft cleared for landing
Actions for Basic Flow	1. Track Aircraft 2. Assess Flight Path 3. Manage Aircraft 4. Track Aircraft
Extension Points	None
Alternate Flow	None

©Figure 5.8 Mapping of Use Case description into basic flow of actions [1]

The information in a Use Case description table is sufficient without further interpretation for mapping into an Activity diagram that shows the flow of actions for the actors and the system. This is depicted in Figure 5.8 [1]. The Activity diagram is for the ATCS (as a black box system) and a compliant aircraft (the actor). As previously noted, the interpretation of Use Case 1 led to three included use cases, each of which are actionable. Thus the diagram on the right side of the figure will need a third Basic Action when domain knowledge is included.

2.2 Functional Allocation to Subsystem Interoperation

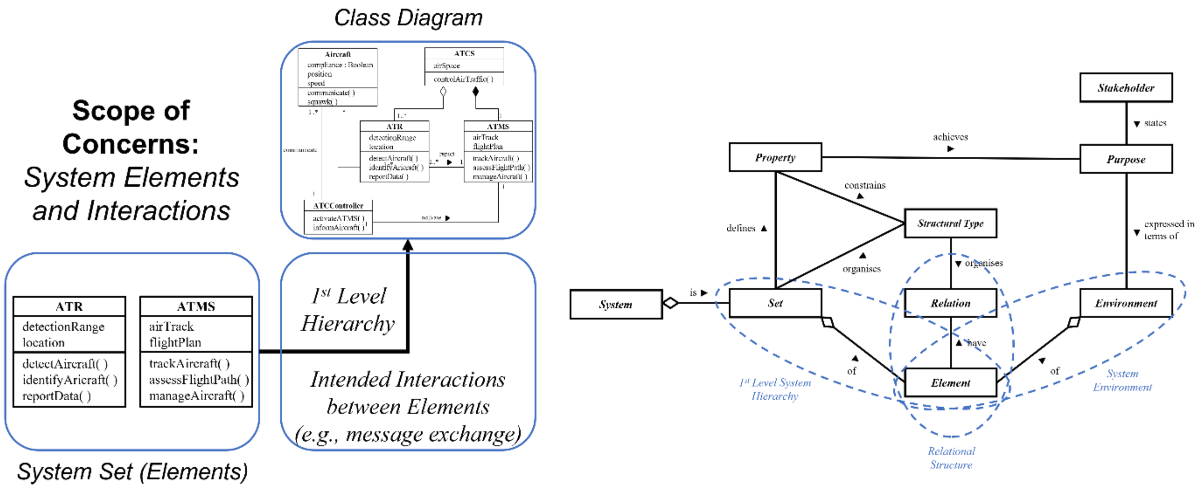
This subsection shows how to transform object-oriented models beginning with functional allocation of basic behaviors represented by Activity Diagrams into Class or Block Diagrams, and concludes with the specification of interoperations that can be synthesized into Sequence Diagrams.



©Figure 6.10 Functional allocation to the system set [1]

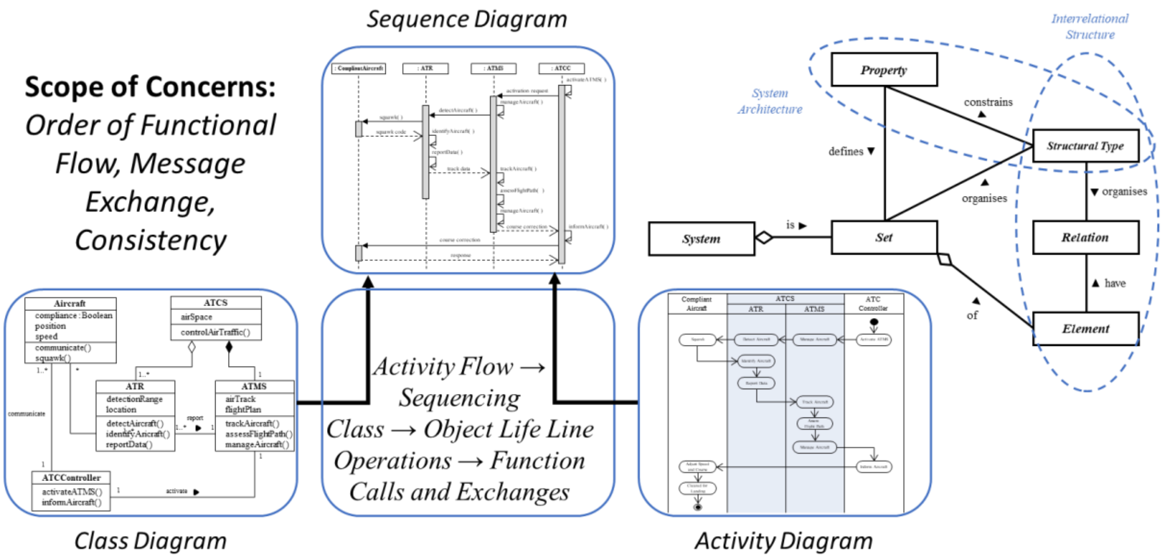
The Activity diagram for the ATCS (as a black box system) is further elaborated (i) to show interactions with an ATC Controller as well as a Compliant Aircraft, and (ii) define system flows of the two ATCS components, an Air Traffic Radar (ATR) and an Air Traffic Management System (ATMS). The integrated model of architecture and system provides a logical structure for understanding the interrelation between the first level system hierarchy represented by a Class or Block Diagram and the system behavior represented by an Activity Diagram.

Figure 6.10 [1] depicts how functional allocation can be accomplished with engineering precision by associating operations of the system elements with actions to be performed by the elements.



©Figure 6.11 Specification of system structure model [1]

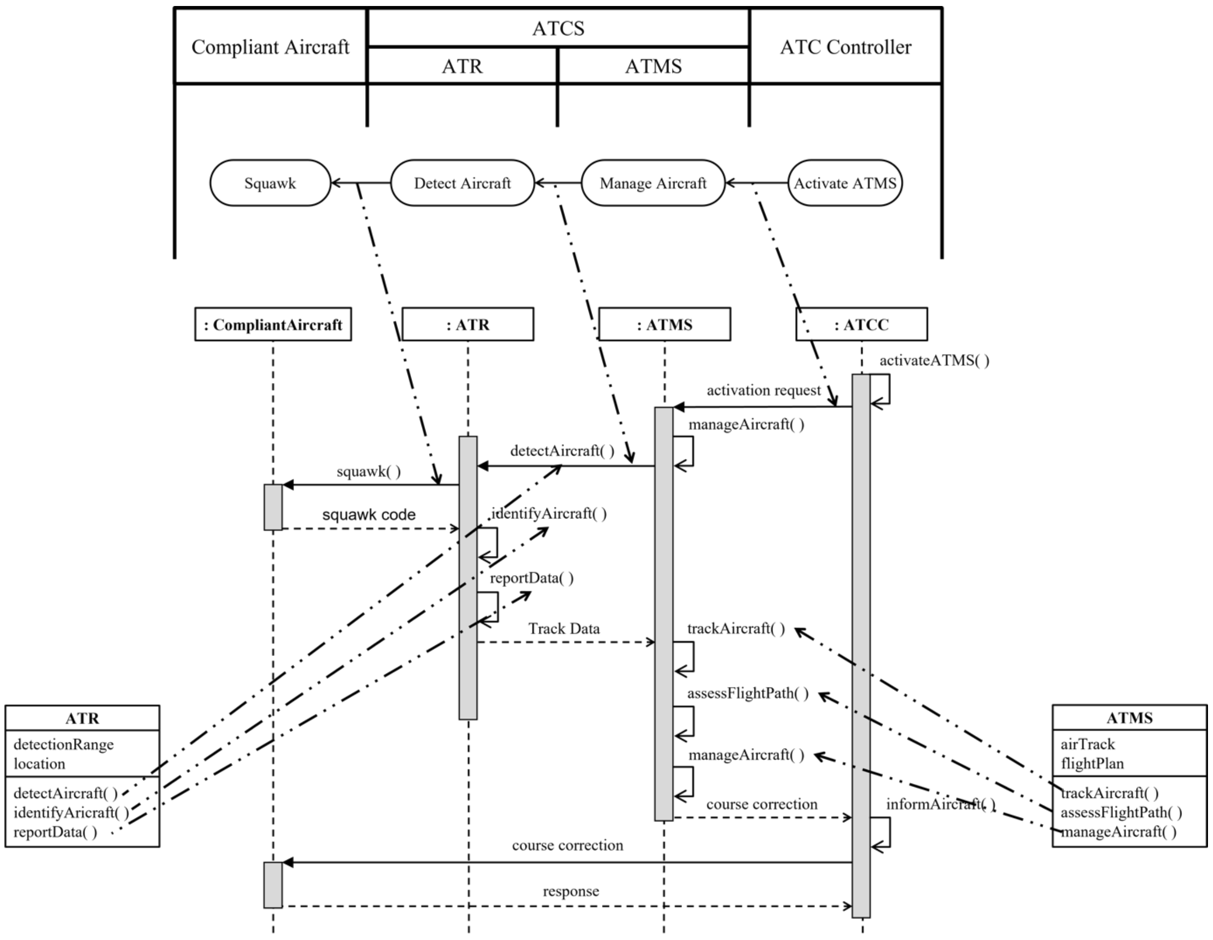
As depicted in the integrated model of architecture and system in Figure 6.11 [1], the interrelations between the 1st level system hierarchy, the system environment, and the relational structure must be understood before intended (as well as inherent) interactions between them can be specified. In the system structure model, the flow of actions will be the basis of specifying intended interactions (e.g., message exchange) between elements.



©Figure 6.12 Synthesis of essential architecture [1]

As depicted in Figure 6.12 [1], in order to specify the system architecture, the Sequence Diagram is the object-oriented model type that provides an interrelational structure within which to realize the myriad of properties defined in the previous diagrams. In terms of object-oriented model elements, the model transformations are realized by the following associations: Class → Objects → Lifelines; Operations → Function Calls. Synthesis and normalization will also be needed.

It is challenging to map the transformation and synthesis in general terms but the example from the ATCS tutorial makes clear the details of how the associations are being realized. These details are depicted in Figure 6.13 [1].

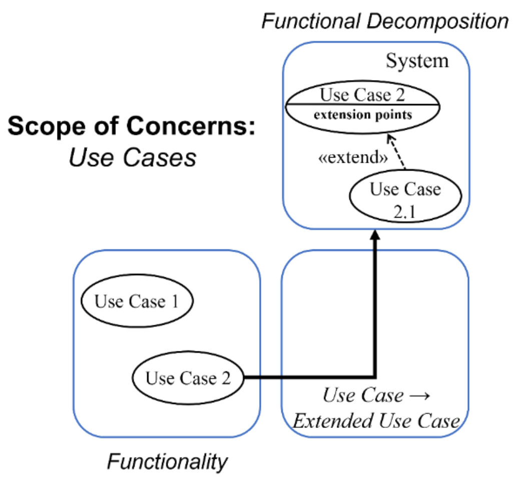


©Figure 6.13 Details of the synthesis transformation [1]

The figure provides details of the synthesis for creating the ATCS Sequence diagram. The actions in the Activity diagram in Figure 6.7 [1] of the tutorial have been associated with the class operations that enable them. The associated flow at the subsystem level of the ATCS represents the internal processes of the ATCS as a system. This flow is now seen to progress downwards through the lifelines of the Sequence diagram.

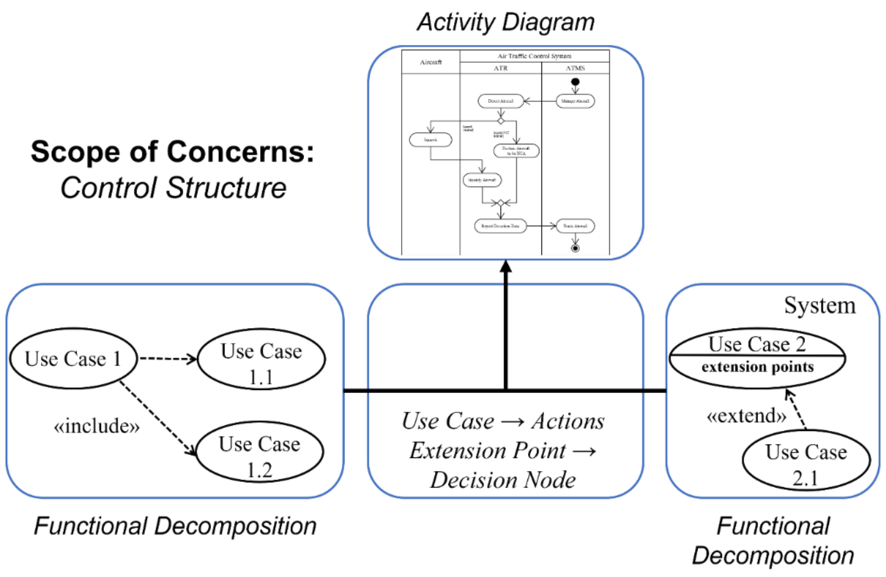
2.3 Functional Decomposition to Alternative Behaviors

The basic behaviors represented by initial Activity Diagrams must be extended to accommodate alternative scenarios. This subsection returns to the Use Case diagram and shows how to transform the object-oriented representations to extend the basic behaviors represented by the Activity Diagrams. Control structures must be defined or revised to accommodate the integration of system behaviors across the scenarios. This requires the ability to combine fragments and create objects in a family of Sequence Diagrams.



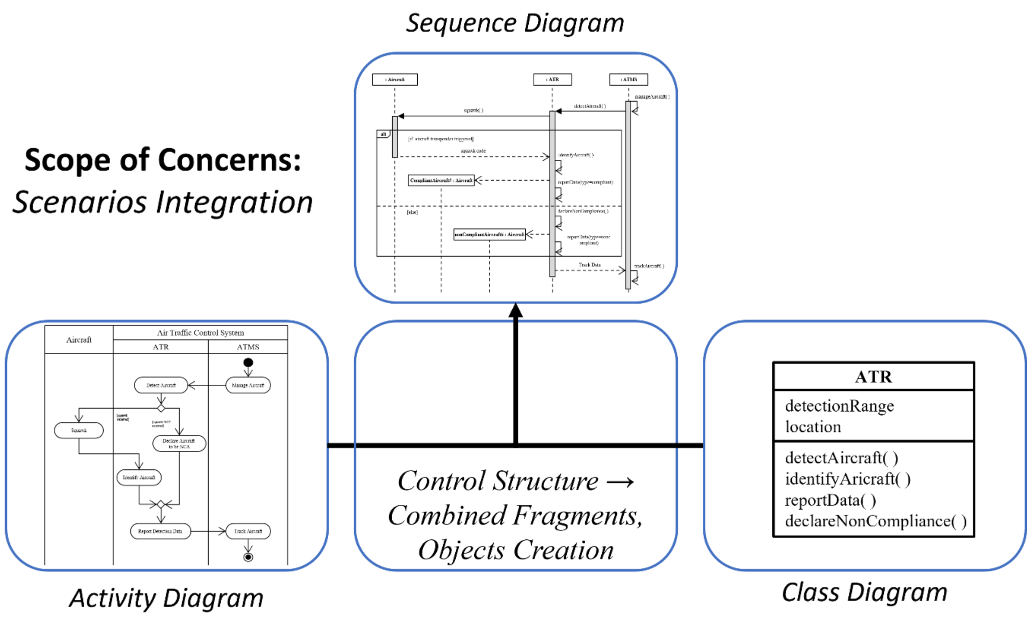
©Figure 7.8 Iterative functional decomposition for alternative behavior [1]

The definition of alternative behaviors begins with extensions of the basic functionality to accommodate alternative scenarios. Similar to the included use cases for functional decomposition, extended use cases provide a representation that supports modelling of alternative behavior. Specific conditions must be met at the extension point in the base use case before the extended use case is invoked. The model specification and transformation are depicted in Figure 7.8 [1].



©Figure 7.9 Specification of integrated behaviors [1]

The structure of the Use Case Description table has provisions for the information and associations contained in Figure 7.9 [1]. If the extended use case is actionable it can be associated with an action in the alternative Activity Diagram. The extension point in the base use case can be associated with a decision node. This will create alternative flows in the Activity Diagram.



©Figure 7.10 Synthesis of essential architecture with alternative behaviors [1]

The complexity of scenario integration can be mitigated by integrating an alternative control structure for each scenario individually into the basic Sequence Diagram. This can evolve the basic control structure into combined fragments using "if, then; else" statements and object creation. A family of alternative diagrams linking each extended use case (associated with an alternative scenario) eventually to an alternative interrelation structure (represented by a Sequence Diagram) can then be specified. Figure 7.10 [1] depicts the transformation for the tutorial example.

3. Future Work

The communities contributing to the development of MBSE have for the past two decades focused on modelling languages, tools, and to some extent methodologies. For example, a significant number of challenges identified in the joint MBSE initiative by INCOSE and OMG have been addressed at various levels of satisfaction through the evolution of standardized languages such as the SysML.

However, it has been observed that in the practical application of MBSE, models are often developed without either explicit acknowledgement or exploitation of mathematical foundations, and the benefits of mathematical precision are either not considered or only considered post model development. An example is using mathematics to demonstrate confidence in the validity and accuracy of models from different modelling teams. This is akin to and as erroneous as treating critical specialist disciplines like system safety as a 'bolt-on' post design item to address their specialist concerns.

Therefore, we propose that the upcoming works of the Mathematical Formalism DSIG should be focused on the following objectives:

1. raising OMG awareness of research results within MBSE communities that establish the benefits of applying mathematical methods and formalisms to support MBSE practices.
2. evolving the model transformation approach presented in this paper to specify formal endogenous and exogenous transformations, thereby enabling consistent model development and model synchronization.

3. advancing the formalisms in this paper to address emerging challenges in the integration of digital engineering and MBSE. A collaboration between the Math DSIG and the Ontology PSIG is considered a critical step for this mission and is planned.
4. collaborating with tool providers in the implementation of model transformations in commercially available tools and standardization of the underlying formalisms via OMG. Successful adoption and deployment of UPR 1.0 demonstrated the feasibility of this specific task.

The details of the above future work will be informed by deriving prioritized needs for the mathematical formalisms, based on continued collaboration with OMG groups and consultation with key players from relevant industries on established engineering practices. The collaboration and consultation will become a regular modus operandi to ensure benefits of the formalisms can be established, evidenced and realized in practice. It is our goal that with successful completion of the above objectives, MBSE practices will shift from being 'largely unaware of mathematical foundations' to 'consciously exploiting mathematics throughout the practice of modelling'.