International Journal of Hybrid Information Technology
Vol.7, No.2 (2014), pp.57-70
http://dx.doi.org/10.14257/ijhit.2014.7.2.07

Drafting Blueprints for Car System Requirements

Sabah Al-Fedaghi

Kuwait University
sabah.alfedaghi@ku.edu.kw

Abstract

Capture, specification, and communication of requirements is now a critical issue in the product development process. Systems Modeling Language (SysML) was developed to address systems engineering needs. In this approach, use cases are developed along with lists of actions necessary to put them into practice. This paper focuses on the early phases of applying textual and diagrammatic narratives in requirements specification and examines a SysML-based representation of a development process as an example. The paper shows that the series of SysML representations lacks a nucleus around which the various phases of the development process can evolve. The paper produces a sample core by using a Flowthing Model (FM) and contrasts it with the multifarious textual and graphical descriptions of SysML to demonstrate the viability of FM for modeling of requirements and design phases.

Keywords: *SysML, product development life cycle, requirements specifications, use case*

1. Introduction and Problem Description

A product development life cycle includes distinct and ordered phases, starting from requirements specification and ending with delivery of the product. In the first phase of the development life cycle, high-level requirements are defined from analysis of the system requirements that do not involve design or verification detail and that may include system constraints. The output of this phase is used to derive design requirements utilized in the implementation phase without further specification.

Capture, specification, and communication of requirements is now a critical issue in the product development process. In this context, representation and management can be plagued with problems because requirements are often ambiguous, incomplete, or contradictory, resulting in profound impacts on the quality and cost of system development. A highly recurrent cause of failure is poor communication of requirements between developers and users [1]. In general, a cause-effect quality-related relationship exists between requirements specification and the final system [1].

In the software development life cycle,

> No other part of the conceptual work is as difficult as establishing the detailed technical requirements... No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later. [2]

Many software failures can be attributed to the inherent complexity of the development process [3]. Studies have shown that the majority of errors are made during requirements analysis, and that most of these errors are not found until the later phases of development [4].

ISSN: 1738-9968 IJHIT

> We are still having difficulties getting the requirements right. We do a poor job of specifying what it is that we want built. Requirements are often ambiguous, unclear, incomplete, or contradictory. And developers often guess what is desired, only to have to come back later in the cycle and rework their software once discrepancies have been discovered. [5]

The cost of fixing a requirements error grows dramatically during the process: correcting an error in a later phase can cost up to 200 times more to fix than one corrected early [6]. This problem in requirements specification is not limited to software:

> Requirements errors are often the most serious errors. Investigators focusing on safety-critical systems have found that requirements errors are most likely to affect the safety of embedded system than errors introduced during design or implementation. ([4]; see references therein)

Current practices in management, control, integration, verification, and validation of requirements may rely on tabular formats and graphical notations [4], *e.g.*, Requirements State Machine Language (RSML) [7], and Specification Toolkit and Requirements Methodology (SpecTRM) [8]. Significant progress has been achieved with the introduction of such methodologies as object-oriented methods [9-11], unified modeling language (UML) [12, 13], and agile development [14]. The resultant tools can be used at different levels of the development phases. For example, Statecharts [15] used to describe the behavior of systems can also be used to specify requirements during design.

UML has gained significant acceptance as a diagrammatic language for specifying, visualizing, constructing, and documenting object-oriented systems development. Systems Modeling Language (SysML) [16-18] is based on UML but addresses systems engineering needs and is more suitable "to analyze, specify, design, and verify complex systems, … to enhance systems quality, improve the ability to exchange systems engineering information amongst tools, and [to] help bridge the semantic gap between systems, software, and other engineering disciplines" [19]. "In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which is used to integrate with other engineering analysis models" [20].

SysML involves modeling of blocks instead of classes in UML, and is said to be easier to learn and apply, adding two new diagrams (requirements and parametric diagrams) to the seven included from UML, for a total of nine diagram types. SysML makes it possible to generate specifications in a single language for heterogeneous teams dealing with the realization of system hardware and software blocks. Knowledge is thereby captured through models stored in a single repository, enhancing communication among and within teams. In the long term, blocks can be reused because their specifications and models enable suitability assessment for subsequent projects [21].

In UML and SysML, the notion of *use case* is most often "associated with requirements" [22]. The modeling process starts by setting a context that includes what is included in the system and what is not, who and what will interact with the system, and what information will be passed to and from the system. Then, use cases are derived, analyzed, and refined by stakeholders and analysts [23]. Accordingly, use case specifications are developed with lists of actions necessary to put them into practice.

> A use case describes a specific operational aspect of the system... It specifies the behavior as perceived by the actors (user) and the message flow between the actors and the use case. An actor may be a person, another system or a piece of hardware external to the system under development... [24]

A use case yields an *observable result* to a particular actor or user [25]. It describes an interaction using text and diagrammatic forms. Alternative paths specify different scenarios the system can follow beyond its typical procedures. Diagrammatic representation can enhance the clarity and hence the understanding of text-based description of requirements, but it does not substitute for text. In general and according to Faulk and Brackett [1], users find diagrammatic representation more acceptable than text and symbols, whereas many developers prefer to work from textual description.

This paper focuses on the early phases of *applying* narratives—textual and diagrammatic—in requirements specification. It focuses on a *sample* SysML depiction of a development process. The paper shows that the sample requirements description is infested with fragmented representations imported from UML. The sequence of SysMl representations lacks a fundamental base or nucleus on which different phases of the development process can be developed, analogous to the central role of the blueprint for a complex project such as a high-rise building, where it serves as the core around which the building's framework, an electrical system, water system, and interior walls will be built by their various specialists. This paper develops an alternative representation by using a tool called the Flowthing Model (FM), then contrasts the result with these multifaceted textual and graphical descriptions, demonstrating the viability of FM in the requirements and design phases as a core or base.

The next section reviews the SysML design process in the context of a sample study case. To provide background for the new methodology, and for the sake of completeness, FM is briefly described in section 3. Section 4 recasts the study case in terms of FM in order to compare the two methodologies side by side.

2. Motivational Study Case: Embedded Systems Requirements

Quadri *et al.* [30] propose use of UML in "Model-Driven Engineering" for system specifications in order to increase comprehensibility as it enables designers to provide high-level descriptions that easily illustrate internal concepts. Their work is within an EU [31] project that aims to utilize model-driven techniques in developing real-time and embedded systems for avionics and surveillance systems. The project proposes using the UML profiles SysML [5] and MARTE [6] to construct tools and technologies that support designing and eventually automatically generating code. Quadri *et al.* [30] aim to develop a design methodology that includes both SysML and MARTE "while avoiding incompatibilities resulting from simultaneous usage of both profiles." They illustrate their methodology by means of a real-life embedded systems case study: a car collision avoidance system.

In the initial specification phase, diagrams are utilized to integrate SysML requirements concepts. Use case Scenarios are then developed, and each use case is converted into a SysML block. Thus, once the functional description is complete, the designer can partition the system to determine which part of the system needs implementation in hardware and which in software.

The study case deals with a car collision avoidance system (CCAS) that serves to detect and prevent collisions with incoming objects. The CCAS contains two types of detection modules:

- A radar detection module emits waves that collide when an incoming object is reflected. The data are sent to an obstacle detection module where the distance to the incoming object is calculated and sent to a primary controller.

- An image tracking module determines the distance to the car by means of image computation. The camera sends the data to a secondary controller, which calculates a distance; if closer than a specified value, the result is sent to the primary controller.

The primary controller acts according to the situation at hand. For an imminent collision, it can carry out certain emergency actions, such as stopping the engine or applying emergency brakes; otherwise, it can decrease the speed of the car and apply normal brakes.

Accordingly, Quadri *et al.* [30] start with requirements specification to describe the initial phase of system conception. Figure 1 shows the different functional requirements: the Global Collision Avoidance Strategy and the derived requirements: the Imminent Collision Strategy, Near Collision Avoidance Strategy, Additional Timing requirements, and Changing Lanes Strategy. These specifications rely on use case scenarios and functional blocks for completion. The use case scenarios are then developed as shown in Figure 2. Once the system requirements and use case scenarios are specified, a functional block description is drawn, as shown in Figure 3.

At this point, Quadri *et al.* [30] discover a need to refine the textual description of Fig. 1 where a related use case scenario and a functional block have to be added. We do not show the new version of that figure.

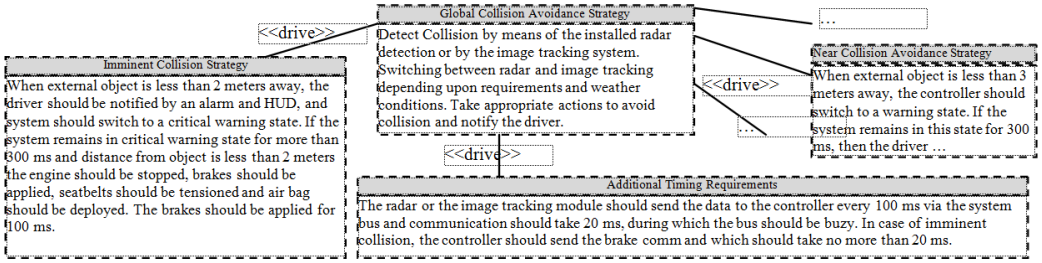


Figure 1. Partial view of different case scenarios related to the project (from [30])

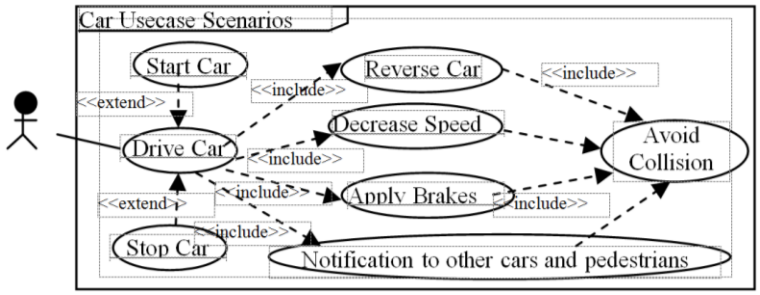


Figure 2. The different case scenarios related to the project (redrawn from [30])

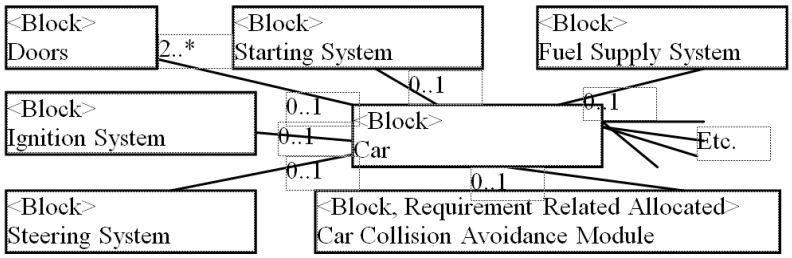


Figure 3. High-level specification using SysML block concepts (partially redrawn from [30])

As claimed previously, we can observe a multiplicity of fragmented representations in SysML imported from UML, including different narratives, diagrams, and notions. The sequence of representations lacks a constant nucleus or base on which to build diverse phases of the process. A brief overview of the alternative tool, FM [26-29], is given in the next section.

3. Flowthing Model

In UML and SysML, "the primary characteristic distinguishing one end of the application spectrum from the other is whether *activities* accept inputs and provide outputs while they are executing" [32] (italics added). *Activity* corresponds to an intuitive notion of items: *things that change*. Thus, an *item* is an entity that can flow through a *system*, *e.g.*, physical matter and objects, energy, data, software objects. An activity is the transformation of items in which items are taken as inputs and provided as outputs.

> Activities are one of three kinds of behavior model in UML, the other two being interactions and state machines. Activities highlight how outputs of one subfunction flow to the inputs of another, while interactions focus on messages between objects, and state machines emphasize object states and transitions between them based on incoming signals. [32]

But a more general conceptualization can be made. Transformation can be related to change in three ways:

1. Change in *sphere* (system, environment) through being released, transferred, and received from one sphere to another.
2. Change in *existence* through being created/re-created, e.g., if a things is deleted, then it is changed.
3. Change in *form* through being processed to change one or more features, e.g., shape, color, size.

Things that change in form, sphere, or existence are called in FM: *things that flow (flowthings)*. Things that flow are things that are transformed in form, sphere, or existence through being created, released, transferred, processed, arrived, and accepted. This characterization is a more general conceptualization of SysML's *things that change* and *transformation from input to output*.

Flow in FM refers to the exclusive (*i.e.*, being in one and only one) transformation among six *states* (also called stages): transfer, process, creation, release, arrival, and acceptance, as shown in Figure 4. We use *Receive* as a combined stage of *Arrive* and *Accept* whenever arriving flowthings are always accepted.

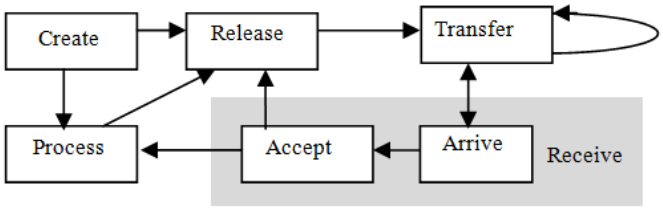


Figure 4. Flowsystem

The fundamental elements of FM are as follows:

Flowthing: A thing that has the capability of being created, released, transferred, arrived, accepted, and processed while flowing within and between "units" called *spheres*.

A flow system (referred to as *flowsystem*) is a system with six stages and transformations (edges) between them. The flow notion in FM corresponds to "atomic flow" in SysML, in which only a single type of input or output is specified. Strangely, flows in SysML can be discrete (flowthings), streaming, or control, but how does "control" flow? It is not a flowthing. In FM, flows can be controlled by the progress (sequence) of a stream of events (create, release, transfer within and between spheres, receive, …), or by triggering that initiates a new flow. For example, in a computer program, control does not flow through instruction; rather, instruction flows through the control sphere.

Spheres and subspheres: These are the environments (constituents) of the totality of the system, such as, e.g., the sphere that comprises a company, a computer, and a person. Spheres and subspheres are the conceptual divisions and subdivisions, each with its boundary, in a decomposition made to manage complexity. A sphere can include the subsphere of a flowsystem as a terminal node in this breakdown.

In UML and SysML, *Control* is the determination of the points at which activities perform their transformations. Similarly, in FM control is the determination of the points at which transferring, releasing, receiving, processing, and creation of flowthings are performed.

Triggering: Triggering is a transformation (denoted by a dashed arrow) from one flow to another, e.g., a flow of electricity triggers a flow of air.

The exclusiveness of FM stages (*i.e.*, a flowthing cannot be in two stages simulaneously) indicates synchronized change of the flowthing. A flowthing *cannot be changed in form and sphere simultaneously*. This is a basic systematic representation of flowthings. Suppose that a message is distorted while being transmitted (change in form and sphere) and then restored at the destination (using, say, parity-bit correction technique). This case can be described in FM as shown in Figure 5, where the white circle represents the original message and the dark one represents the distorted message. There is no change in form when moving from one sphere to another. Also, the change in form (white/dark, dark/white) occurs while not in the transfer stage.

Similarly, no flowthing can be created and change form (be processed) simultaneously. Creation means coming into existence; hence, this transformation is in conflict with change in

form (being processed) since the latter requires pre-existence. Also, a flowthing cannot be transferred and arrive at the same point in time, and it cannot be received and processed instantaneously.

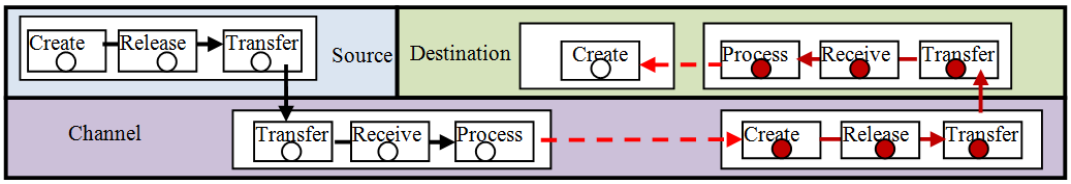


Figure 5. Message is exclusively in a single stage

The issue remains of being released and transferred exclusively. The point of *transfer* is the point of departure from the sending sphere. Such a moment is surely preceded by a decision to output the flowthing. This decision is actually materialized in time and not a strike to the transfer stage. For example, even in a human being, there is an instance of intent between mental release of a concept (conscious thought) and actual transfer of the concept in a spoken or written communication. Thus a "state" exists that precedes actual transfer no matter how insignificant the time between the decision to release and its implementation by transfer. This is the state of the change from resident item to exported item before actual exit from the source sphere; however, this release state is a significant stage in many situations; *e.g.*, a human being prepares a speech, products are stocked in inventory for shipping, *etc.*

4. Contrasting the Two Representations

In UML 2, activity diagrams represent activity by a round-cornered rectangle as shown in Figure 6, where "Item flow is omitted for brevity" [32]. The arrows in the figure are "control flow lines." Figure 7 shows the corresponding FM representation, where the conceptual picture is more complete.

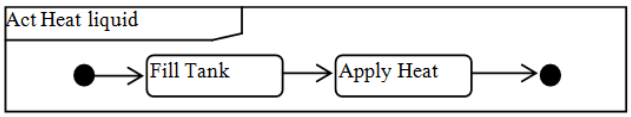


Figure 6. Activity definition (from [32])

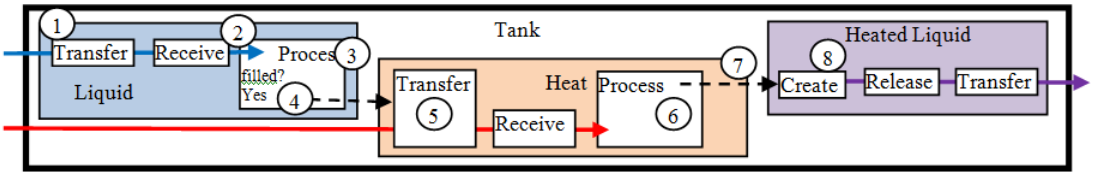


Figure 7. FM representation that corresponds to Figure 6

The Tank sphere has three subspheres: Liquid, Heated Liquid, and Heat. *Sphere* here denotes a conceptual system that contains other subspheres, or it is a flowsystem. What is Tank? It is a sphere that ecompasses Liquid, Heated Liquid, and Heat; analogous to the notation *Tank(Liquid, Heated Liquid, Heat)*. What is Liquid? It is a sphere of type flowsystem where the flowthing *liquid* flows through its stages of transfer, receive, and process. Transfer (circle 1) occurs at a "valve" where the liquid enters (this is deduced from the direction of the

arrow) and is received (2). The Process stage (3) involves checking the liquid level. If the result of this checking process indicates the tank is filled, it triggers (4) a flow of heat to the tank (5) to be received and processed (6). Processing the heat (*i.e.*, heating the surroundings in the tank) triggers (7) the creation (generation) of Heated Liquid (8). Note that this conceptual *creation* means the appearance of this flowthing for the first time. This *creation* is analogous to the notion of plastics being *created* when polymers are added to a resin, or, more dramatically, the *conceptual* creation of Mr. Hyde from Dr. Jekyll, where Mr. Hyde appears in the sphere of Dr. Jekyll from nowhere; he has been created—not arrived from another sphere—otherwise he would have been transferred and received from a source outside the sphere.

In Bock's [32] description, the activity notation is based on Activity definition, Usage, and Execution. For example, Figure 8 shows an activity with the same item definition ("classifier" in UML), Water, used as output from different activities, Heat Liquid and Supply Water, and input to the same activity, Dispense Water [32]. The arrows in Fig 8 now have double meanings: some represent control flow, and some denote item flow. The figure can be contrasted with the FM representation shown in Figure 9. The figure represents the flow of Water (1), Liquid (2), and Heat (3) to produce Hot mix (4). Some Water flows directly as Dispense water (5). The Hot mix also generates Dispense water (6).

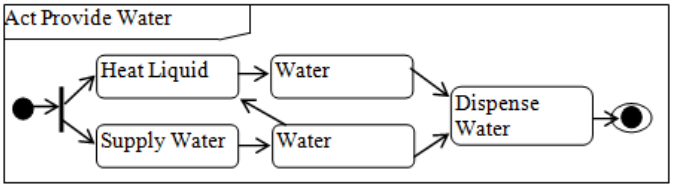


Figure 8. Multiple items of same kind input to an activity usage (from [32])

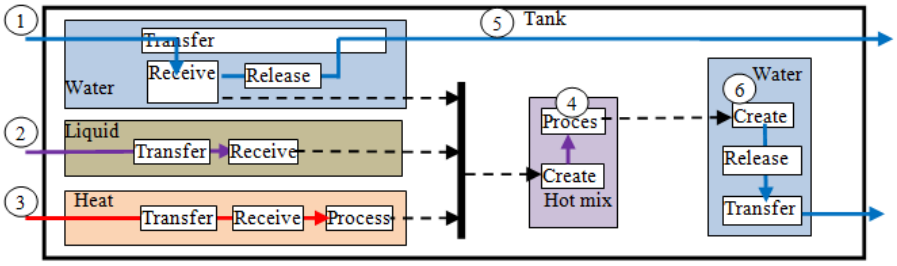


Figure 9. FM representation

Comparing the two diagrammatic representations, FM appears as a straightforward description of the situation, similar to textbook illustrations in, say, Physics, while the activity diagram provides a shorthand sketch of the situation. Note that all types of notions can be superimposed on the basic FM description, *e.g.*, logic, synchronization, constraints symbols. In Figure 9, the vertical bar is taken from Petri nets notation to indicate "firing" (triggering to generate Hot mix) that depends on the presence of Water, Liquid, and Heat. Note also that Hot mix is not present in Figure 8, but deduced and added into the FM description by the semantics of events. If heated liquid is supplied with water, a Hot mix of liquid and water is generated, causing the flow of Dispense water. This continuity of narration is "forced" by basing FM on the concept of conceptual flow of flowthings.

Bock [32] enhances Figure 8 to permit distinguishing the temperature of the Dispense water as Hot or Cold:

This requires item usages on the activity definition (parameters in UML)… Parameters … to items flowing in and out of activities. Parameters are named to distinguish each input, … This requires another layer of item usage, the item usages on activity usages.

Accordingly, Figure 8 is modified as seen in Figure 10.

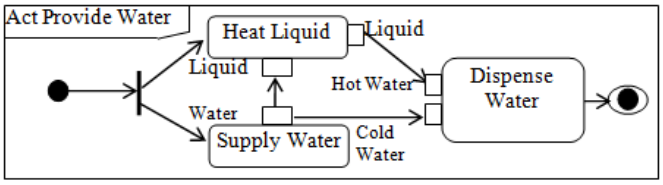


Figure 10. Use of pins (from [32])

In FM such additional requirements can be incorporated into the conceptual description without the need for additional notions such as parameters and pins. Figure 11 shows the required modifications. Cold water (1) and Hot water (2) are now viewed as separate flowthings (different types—colors—of arrows) with their distinguished streams of flow.

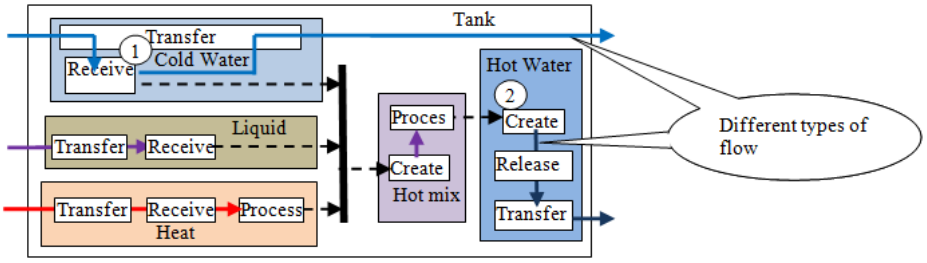


Figure 11. FM representation with Cold and Hot water

Bock [32] then describes control capabilities in UML 2, and SysML, *e.g.*, needed for disabling streaming activities as when a driver turns off a car, or to control buffering. Control is treated in SysML as if it were an item accepted as input or provided as output through parameters. It supports control values and operators with a classifier called CONTROLVALUE. The classifier has instances ENABLE and DISABLE. These values can flow through an activity execution like any other item [32].

In FM, Control is an integral part of the conceptual representation. Figure 12 shows the addition of such a feature. The hot mix triggers (1) temperature measurement that is created (2) and processed (3). If it is below or over the threshold (4 and 5, respectively), adjustment in the flow of heat is triggered (6).

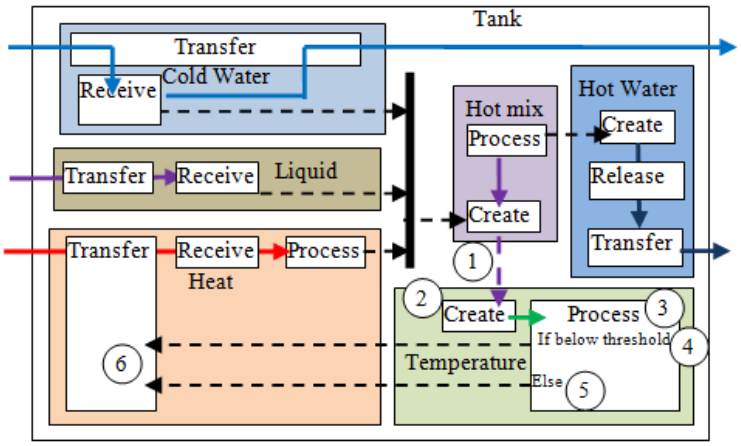


Figure 12. In FM, Control is an integral part of the total conceptual view

5. Model Embedded Systems Requirements Revisited

The aim of this section is to develop a conceptual representation that can serve as a nucleus for further development; however, only a partial representation can be shown because of space limitations. We start with a partial view (Figure 13) of the Car Use case Scenarios shown in Figure 2.

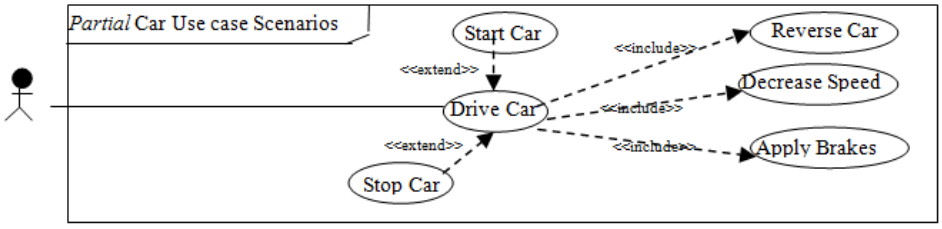


Figure 13. Partial view of case scenarios related to the project shown in Figure 2

The figure includes the cases: Drive Car, Start Car and Stop Car and includes subcases or Use case Scenarios: Reverse Car, Decrease Speed, Apply Brakes. This partial view describes the main components of the car system over which other subsystems can be superimposed.

Note that the figure presents a fragment of the description of the car system. Other fragments are provided by text, high-level specification using SysML block concepts (shown partially in Figure 7), and other diagrams. It can be pointed out here that the "embedded system for avionics and surveillance" is built over the basic description of the car system, and this is what these SysML descriptions try to represent in a piecemeal manner. This system includes Drive, Start, Stop car, Decrease speed, Apply brakes, and so forth; accordingly, we utilize these sketchy terms and words appearing in SysML diagrams to build a broad car system that can be understood by stakeholders and form a core for communicating other requirements including collision avoidance.

Figure 14 shows the FM representation of a sample car system. Note that for simplicity's sake, several assumptions are made such as that the car has three gear positions: neutral, drive and reverse. In a less compacted representation, a full conceptual map could be developed to reach the level of detail shown in typical engineering schemata. The FM description can be generated to any level of detail, just as with the blueprint of a building in which, say, some interior partitions are marked but not completely represented in the drawing. Figure 14 presents a compact and convenient level of description; a more complete representation can be developed in real cases.

The driver in Figure 14 uses his/her key to create a signal (circle 1) that flows (2) to the car system to be processed (3). There, if the gear is in the neutral position (4), the engine turns on (5). Note that both conditions should be present:

A signal from the driver AND a signal from the gear that it is in the neutral position.

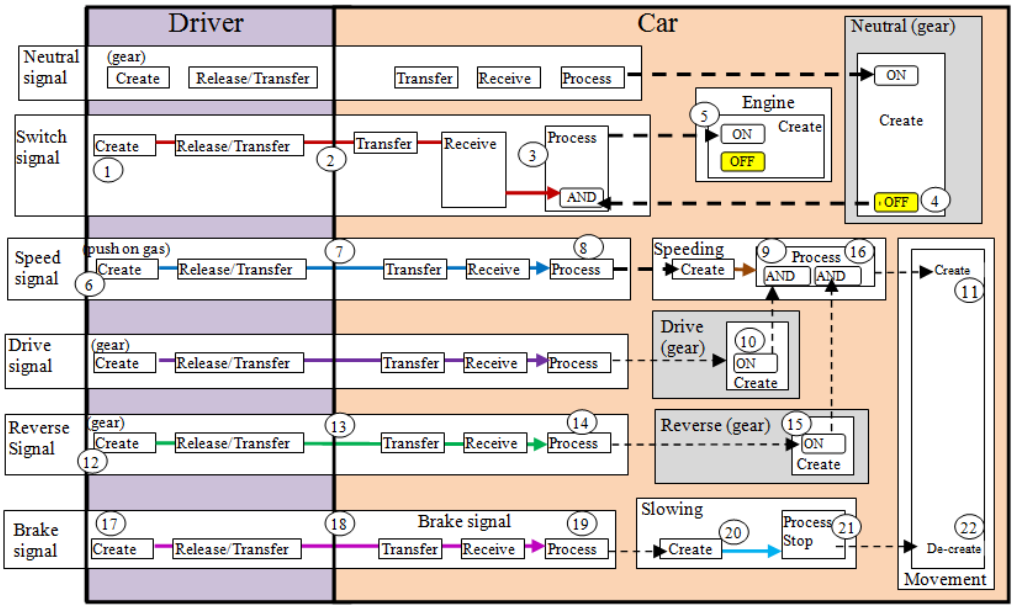


Figure 14. A simplified FM description of the car system

This last condition is modelled as the gear triggering the processing (3). Triggering in this case can be implemented by a signal from the gear informing of its position.

Speeding starts with the driver creating (6) a signal (pushing down the gas pedal) that flows to the car system (7), where it is processed (8). There, the speeding mechanism is triggered (9); however, the gear in this case should be in the drive position to create forward movement (11). Similarly, for reverse movement, the driver first creates the reverse signal (12) that flows to the car system (13), where it is processed (14). There, if the gear is in the reverse position (15) AND there is a drive signal (16), this causes the car to move in the reverse direction.

When the driver creates the brake signal (17), it flows (18) to the car system, where it is processed (19) to trigger the creation of the mechanism of slowing (2). The slowing is processed at different degrees (21) to trigger gradual stopping of the car and cessation of movement (22).

6. Conclusion

This paper focuses on the piecemeal, heterogeneous conceptual descriptions adopted in SysML. One disadvantage of such an approach is the lack of a core specification that plays the role of central reference in system specification. An alternative representation is presented and applied to published SysML cases. We can conclude that the proposed Flowthing Model presents a viable tool for creating a core schema for describing requirements. Further research will reveal how it can be integrated into the project development life cycle.

Acknowledgements

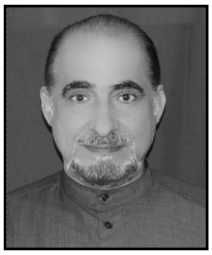
I would like to acknowledge the reviewers, as their comments have been constructive and helpful. Also, I would like to thank SERSC for providing a free processing charge for publication.

References

[1] S. Faulk, J. Brackett, P. Ward and J. Kirby Jr., "The CoRE Method for Real-Time Requirements", IEEE Software, vol. 9, no. 5, **(1992)** September, pp. 22-33.
[2] F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", Computer, vol. 20, no. 4, **(1987)** April, pp. 10-19.
[3] J. Chen, H. Wang, Y. Zhou and S. D. Bruda, "Complexity Metrics for Component-based Software Systems", JDCTA: Int. J. of Digital Content Tech. and its Applications, vol. 5, no. 3, **(2011)**, pp. 235-244.
[4] U.S. Department of Transportation, "Requirements Engineering Management Findings Report", DOT/FAA/AR-08/34, **(2009)** May, http://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-34.pdf.
[5] J. Marasco, "Software development productivity and project success rates: Are we attacking the right problem?", **(2006)**, http://www.ibm.com/developerworks/rational/library/feb06/marasco/.
[6] B. Boehm, "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, NJ, **(1981)**.
[7] N. Leveson, M. Heimdahl, H. Hildreth and J. Reese, "Requirements Specifications for Process-Control Systems", IEEE Transactions on Software Engineering, vol. 20, no. 9, **(1994)** September, pp. 684-707.
[8] J. Howard, "Preserving System Safety across the Boundary between System Integrator and Software Contractor", Proceedings of the SAE World Congress, paper 2004-01-1663, Detroit, MI, **(2004)** March.
[9] P. Coad and E. Yourdon, "Object-Oriented Design", Youdon Press, Englewood Cliffs, NJ, **(1991)**.
[10] I. Jacobson, "Object-Oriented Software Engineering", ACM Press, New York, **(1991)**.
[11] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, "Object-Oriented Modeling and Design", Prentice-Hall, Upper Saddle River, NJ, USA, **(1991)**.
[12] I. Jacobson, G. Booch and J. Rumbaugh, "The Unified Modeling Language User Guide", Addison-Wesley, Reading, MA, USA, **(1999)**.
[13] M. Fowler, "UML Distilled, A Brief Guide to the Standard Object Modeling Language", Third Edition, Addison Wesley, Reading MA, **(2003)** September.
[14] J. A. Highsmith, "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems", Dorset House, New York, USA, **(2000)**.
[15] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, vol. 8, no. 3, **(1987)**, pp. 231- 274.
[16] F. O. Hansen, "SysML – a modeling language for systems engineering", **(2010)**, http://staff.iha.dk/foh/Foredrag/SysML-SystemEngineering-DSFD-15-03-2010.pdf.
[17] S. Friedenthal, A. Moore and R. Steiner, "A Practical Guide to SysML: The Systems Modeling Language", Elsevier, ISBN 0123852064, 9780123852069, **(2011).**
[18] T. Weilkiens, "Systems Engineering with SysML/UML: Modeling, Analysis, Design", Elsevier, ISBN: 9780123742742, **(2008).**
[19] Object Management Group UML channel, UML & SysML modelling languages: Expertise and blog articles on UML, SysML, and Enterprise Architect modelling tools, **(2013)** February, http://www.umlchannel.com/en/sysml.
[20] OMG Systems Modeling Language, The Official OMG SysML site, http://www.omgsysml.org/.

[21] Guillaume Finance, SysML Modelling Language explained **(2010),** http://www.omgsysml.org/SysML_Modelling_Language_explained-finance.pdf.
[22] A. Cockburn, "Writing Effective Use Cases", Addison-Wesley, Boston, MA, **(2001)**.
[23] J. A. Lane and T. Bohn, "Using SysML modeling to understand and evolve systems of systems", Systems Engineering, vol. 16, no. 1, **(2013)**, pp. 87–98.
[24] H. -P. Hoffmann, "Systems Engineering Best Practices with the Rational Solution for Systems and Software Engineering", IBM Software Group, Deskbook Release 3.1.2, **(2011)** February.
[25] I. Alexander and T. Zink, "An Introduction to Systems Engineering With Use Cases", IEEE Computer and Control Engineering, vol. 13, no. 6, **(2002)** December, pp. 289–297.
[26] S. Al-Fedaghi, "Conceptual Software Testing: A New Approach", Int. Review on Computers and Software, vol. 8, no. 8, **(2013)** August.
[27] S. Al-Fedaghi, "Flow-based Enterprise Process", Int. J. of Database Theory and Application, vol. 6, no. 3, **(2013)**, pp. 59-70.
[28] S. Al-Fedaghi, "A Method for Modeling and Facilitating Understanding of User Requirements in Software Development", Journal of Next Generation Information Technology, vol. 4, no. 3, **(2013)**, pp. 30-38.
[29] S. Al-Fedaghi, "Schematizing Proofs based on Flow of Truth Values in Logic", IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2013), Manchester, UK, **(2013)** October 13-16.
[30] I. R. Quadri, A. Sadovykh and L. S. Indrusiak. "MADES: a SysML/MARTE high level methodology for real-time and embedded systems", In Proceedings of the 2012 Embedded Real time Software and Systems Conference, Toulouse, France, **(2012)** February.
[31] A. Bagnato, *et al.*, "MADES: Embedded systems engineering approach in the avionics domain," in First Workshop on Hands-on Platforms and tools for model-based engineering of Embedded Systems (HoPES), Paris, France, **(2010)** June, pp. 5-9.
[32] C. Bock, "SysML and UML 2 Support for Activity Modeling", Syst. Engineering, vol. 9, no. 2, **(2006)** May, pp. 160-186.

Author



Sabah Al-Fedaghi

He holds an MS and a PhD in computer science from the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, and a BS in Engineering Science from Arizona State University, Tempe. He has published two books and more than 180 papers in journals and conferences on software engineering, database systems, information systems, computer/information privacy, security and assurance, information warfare, and conceptual modeling. He is an associate professor in the Computer Engineering Department, Kuwait University. He previously worked as a programmer at the Kuwait Oil Company, where he also headed the Electrical and Computer Engineering Department (1991–1994) and the Computer Engineering Department (2000–2007).

