International Journal of Control and Automation
Vol.7, No.4 (2014), pp.125-136
http://dx.doi.org/10.14257/ijca.2014.7.4.12

An Alternative Approach to Multiple Models: Application to Control of a Production Cell

Sabah Al-Fedaghi

Kuwait University
sabah@alfedaghi.com

Abstract

An information system model represents a business establishment and reflects the realities of its organization and operations. One begins to build such a model by drawing a conceptual picture of the establishment as part of its real-world domain. Object-oriented methods and languages (e.g., UML, SysML) are typically used to describe the system at this level. The resultant representation includes several textual and diagrammatic descriptions and is completely heterogeneous in form, with several different conceptual bases. This heterogeneity and multiplicity of models has caused problems that demonstrate the need for consistency between a UML use case model and its corresponding set of textual descriptions. This paper contrasts that approach with a method that provides a single, integrated graphic model that incorporates function, structure, and behavior into an underlying conceptual representation. This model is demonstrated through samples in which UML and SysML techniques were previously applied to a database analysis workflow and control of a production cell.

Keywords: *Information system model; UML; use case diagram; multiplicity of models*

1. Introduction

An information system (IS) model depicts a business establishment and reflects the realities of its organization and operations. Building such a model begins with the drawing of a conceptual (technology-neutral) portrayal of the establishment as part of a real-world domain. It serves as a communication tool among stakeholders involved in building the system and as a guide for the subsequent IS design phase that involves a description of the software system under development. This requirements-capturing phase is an important issue in software engineering that has been extensively researched.

Object-oriented methods and languages (*e.g.*, UML) are typically used to describe the system at this level. Researchers have proposed extending object-oriented software design languages such as UML so they can also be applied at the conceptual level (*e.g.*, [1]). UML 2.0 includes 13 types of diagrams, an increase from the 9 found in UML 1.0, mostly to depict higher-level views of the system and to enhance component-based development [2]. According to Evermann [1], “UML is suitable for conceptual modelling but the modeller must take special care not to confuse software aspects with aspects of the real world being modelled.”

The remarkable aspect of this approach is the multiplicity of models needed in UML, a known problem [3] that contrasts with simply providing a single, integrated diagrammatic model that incorporates function, structure, and behavior [4]. The representations in UML are completely heterogeneous in form, with several different conceptual bases. Use cases are narratives, use case diagrams are sketchy, sequence diagrams reflect the nature of exchange/communication, and so on. The purpose of this heterogeneity is to provide a wide range of options for expression, depending on the situation. This need for multiple models goes beyond only UML diagrams, e.g., Shoval and Kabeli [5] propose a merger of data flow diagrams, entity relationship diagrams, and object-oriented constructs.

This approach of a multiplicity of models in UML has caused several problems. For example, it is difficult to achieve consistency between a UML use case model and its corresponding set of textual descriptions, which are the written explanations of the use case relationships contained in the UML model [6]. "This is a non-trivial problem, because ensuring consistency between a UML model and the textual use case descriptions requires a certain degree of formality in the textual descriptions" [6].

This paper deals with an instance of continuing attempts to supplement diagramming methods with one another, apparently to compensate for a sense of inadequate representation of higher-level views of the system. It is here proposed that it is time to adopt a new paradigm:

Either attempt to go back to the approach of a single, integrated graphic model that incorporates function, structure, and behavior, or attempt to develop a proposed new model that might provide an underlying integrated model for UML representation.

The paper suggests such an integrating model without, at this stage, trying to make a case for either of the two proposals above. The paper introduces a new diagrammatic model that seems to absorb some mixture of the proposed combinations of UML diagrams. To demonstrate the viability of this new conceptual model, called the flowthing model (FM), the paper contrasts a representative example of UML diagrams described by Wang [7] with the proposed methodology with the aim of comparing the two representations side by side. The reader can then validate the claim of this paper that the new method is viable as a tool for modeling requirements at the conceptual level; that is, it is a representation of requirements that is independent of implementation.

2. UML-based Framework

Wang [7] proposes "a framework that applies UML components to ORDBs [Object Oriented Databases] development workflows." It is based on a United Process [8], where UML use cases are utilized to capture the functional requirements in an iterative process that "takes a set of use cases from requirements all the way through implementation, test and deployment" [7]. To illustrate how to apply UML techniques to an ORDBs analysis workflow, Wang [7] begins with development of use case narratives.

> Based on use case narratives, use case diagrams are developed to illustrate the system functionality visually. Both sequence and class diagrams can be derived from the UML use case. The sequence diagram helps the use case generate the class diagram during the analysis workflow. [7]

As shown in Figure 1, a "Place order" example is developed based on a use case narrative. A use case diagram (Figure 2) is also presented to visually illustrate system functionality. Both sequence (Figure 3) and class diagrams can be derived from this UML use case.

> UML offers great potential for ORDBs data modeling. The Version 2 of the UML defines more than a dozen of diagrams, such as use case diagram, class diagram, activity diagram, sequence diagram, state diagram and communication diagram. These diagrams are ideal to

represent multiple perspectives of ORDBs by providing difference types of graphical diagrams during different stages of ORDBs development. [7]

Use case: "Place Order"
Actor: Customer
Precondition "Customer has invoked "Browse catalog"
Flow of Events
1. Customer adds additional item to shopping cart
2. Customer selects "Review order"
3. Customer receives calculation of total cost, tax, and shipping charges
4. Customer selects "Place order"
5. Customer clicks "Submit Order" to update the system
6. Customer's receipt is created

Post Condition: Customer leaves the screen or Web site

Figure 1. Use case narrative: Place Order

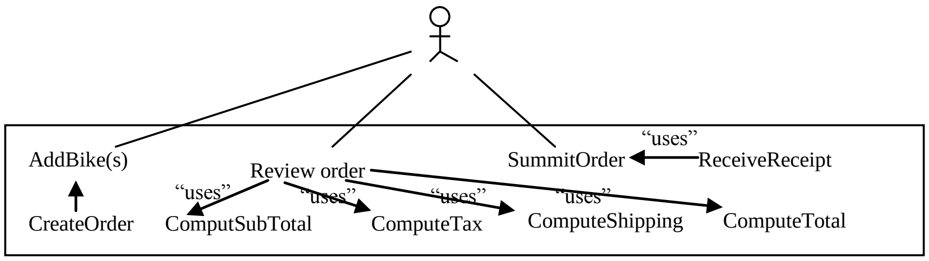

Figure 2. Use case diagram: Place Order

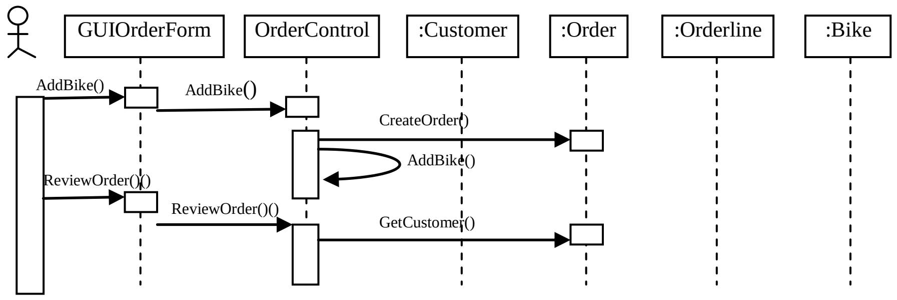

Figure 3. Partial view of sequence diagram: Place Order

Class names can be developed from a list of nouns identified, including actors from use case diagrams that are class names, by checking which use case has what classes and verifying "which classes are responsible for creating, retrieving, updating and deleting methods" [7].

The general claim of the present paper is that problems related to development of representations of IS requirements ought to be addressed on the basis of a new paradigm based on the notion of flow. The feasibility of the new paradigm is demonstrated by recasting Wang's [7] sample system of use cases in terms of a new proposed methodology.

3. Flowthing Model

For the sake of a self-contained paper, this section briefly describes FM, on which the new representation of the sample case is built. FM has been utilized in many applications [9-12]. The example constructed at the end of the section is a new contribution.

The Flowthing Model (FM) was inspired by the many types of flows that exist in diverse fields, such as, for example, supply chain flow, money flow, and data flow in communication models. This model is a diagrammatic schema that uses “flow things” (referred to as flowthings) to represent a range of items that can be, for example, data, information, or signals. FM represents processes using “flow systems” (referred to as flowsystems) that include six stages, as follows:

- Arrive: a flowthing reaches a new flowsystem (*e.g.*, a buffer in a router)

- Accepted: a flowthing is permitted to enter the system (*e.g.*, correct address for a delivery); if arriving flowthings are also always accepted, Arrive and Accept can be combined as a Received stage.

- Processed (changed): the flowthing passes through some kind of transformation that changes its form but not its identity (*e.g.*, compressed, colored)

- Released: a flowthing is marked as ready to be transferred (*e.g.*, airline passengers waiting to board)

- Created: a new flowthing originates (is created) in the system (*e.g.*, a data-mining program generates the conclusion *Application is rejected* as input data)

- Transferred: the flowthing is transported somewhere outside the flowsystem (*e.g.*, packets reaching ports in a router, but still not in the arrival buffer).

These stages are mutually exclusive, *i.e.*, a flowthing in the Process stage cannot be in the Created stage or the Released stage at the same time. An additional stage of Storage can also be added to any FM model to represent the storage of flowthings; however, storage is not a generic stage, because there can be stored processed flowthings, stored created flowthings, and so on. Hereafter, a thing means a flowthing.

Figure 4 shows the structure of a flowsystem. A flowthing is a thing that has the capability of being created, released, transferred, arrived, accepted, or processed while flowing within and between systems. A flowsystem depicts the internal flows of a system with the six stages and transactions among them.

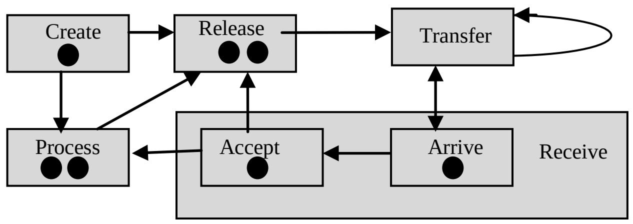

Figure 4. Flowsystem, assuming that no released flowthing is returned. The dark dots denote things at different stages of the flowsystem. The figure can be considered like net marking (instantaneous location of all tokens in the net; Petri net terminology)

FM uses the following basic notions:

Flowthings*:* A thing that has the capability of being created, released, transferred, arrived, accepted, and processed while flowing within and between ‘units’ called *spheres*. Flowthings can be concepts, actions, or information. Information communication involves creating, releasing, transferring, receiving, and processing of information.

The control of the movement of flowthings is assumed to be embedded in the stages; *e.g.*, in Process: *if a flowthing satisfies a certain condition, then it flows to Release*. In principle, no difficulties should exist in conceptualizing such control on the edges, in the manner of Petri nets.

Spheres and subspheres*:* These are the environments of the flowthing. A sphere can have multiple flowsystems in its construction, if needed. A sphere can be an entity (*e.g.*, a hospital and the departments within it; a person or class of persons, *e.g.*, nurses; a computer, a component, and so forth), a location (laboratory, waiting room), communication media (channel, wire), … A flowsystem is a subsphere that embodies the flow; it itself has no subsphere.

Triggering*:* Triggering is an activation (denoted in FM diagrams by a dashed arrow) of one flow by another. It is dependent on flows and parts of flows. A flow is said to be triggered if it is created or activated by another flow, *e.g.*, a flow of electricity triggers a flow of heat, or if it is activated when a condition in the flow is satisfied, *e.g.*, processing of records x and y triggers the creation of record z in the flowsystem of records. Triggering can also be used to start events, *e.g.*, turning on a flowsystem by using a remote signal.

A flowsystem may not need to include all the stages; for example, an archiving system might use only the stages Arrive, Accept, and Release. Multiple systems captured by FM can interact with each other by triggering events related to one another in their spheres and stages.

4. FM Approach to Databases Development Workflows

Wang [7]’s “Place order” example is diagrammed as shown in Figure 5 with an attempt to preserve the semantics of the example as understood in the original paper. Some modifications will be made to the original scenario because the FM account reveals some conceptual gaps in the sequence of events shown in the original UML use case. Initially, in the FM representation, the company sphere comprises two subspheres that are explicitly distinguished: the Ordering department and the Shipping department.

In Figure 5, a catalog is retrieved (circle 1) by the system and released (2) to flow to the customer (3), where it is received and processed (4), e.g., selection and addition of elements ordered from the menu. This processing triggers (5) the creation (6) of ordered elements, an order that flows to the company system (7), where it is processed (8) to trigger (9) a Review of items (10). The Review flows (11) to the customer, where it is processed (12) to trigger (13) the creation of not approved (14) or approved (15). Not approved triggers the process to start again by sending another catalog to the customer (16). An approval note flows to the company (17) where it is processed (18) to trigger the creation of a cost quote (19) that flows to the customer (20). Upon receiving the cost quote (21), the customer may approve (22) or not approve it (23). In case of approval, an indication flows to the company (24) where it is processed and triggers creation of the final order (25). This order flows to the Shipping department (26) where it is processed (27) to trigger:

- Retrieval of the ordered items from stock (28) and sending to the customer (29)

- The creation (30) and sending of a receipt to the customer (31).

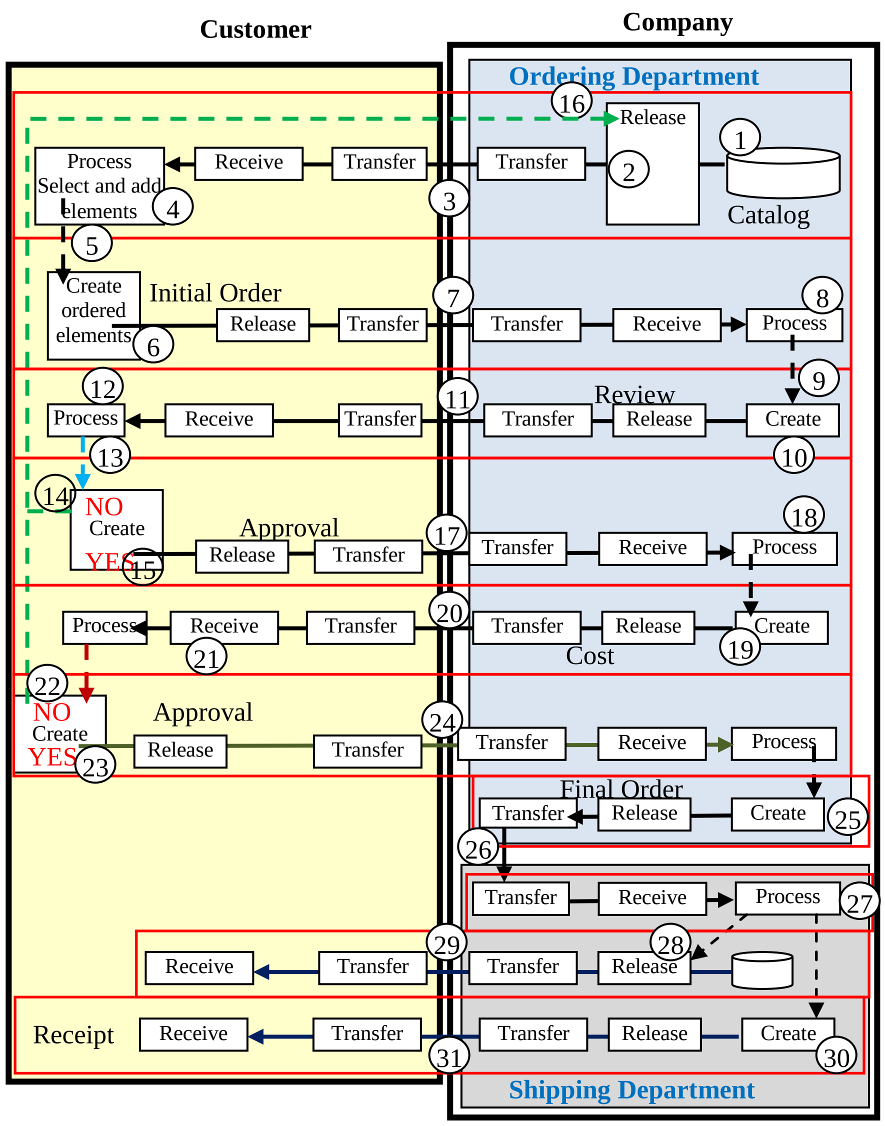

Figure 5. FM representation of Place Order

As shown in Figure 5, the FM representation reflects repeated application of the five generic operations: transfer, receive, process, release, and create. It is characterized by a continuity of events that ensures coherence and completeness in the semantics involved. In Figure 5, several additions are inserted to fill conceptual gaps in the original use case description. For example, approvals have been added in the case of Review and Cost invoice. It is clear that delivering goods is performed by a different entity in the company; thus a Shipping Department has been added.

4. Control of a Production Cell

Systems Modeling Language (SysML) [13] was created using UML's profile mechanism, which represents a subset of UML 2 with extensions [14-16]. "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" [17]. It is designed to support development stages including the specification, analysis, design, and validation of systems engineering applications.

The features that are inadequately included in current architectures of SysML include two types of diagrams that do not exist in UML 2: the requirements diagram and the parametric diagram. Activity, internal block, and block definition diagrams in SysML are modified activity, collaboration, and class diagrams in UML 2, respectively. Sequence, state machine, use case, and package diagrams in SysML are imported directly from UML 2.

Nejati *et al.* [18] address the problem of traceability from requirements to design in terms of expressing requirements as unrestricted natural language statements and the design using Systems Modeling Language (SysML) [4]. According to Nejati et al. [18],

- SysML expresses systems engineering semantics better than UML.
- SysML has built-in cross-cutting links for interrelating requirement and design elements.

Nejati *et al.* [18] use a fragment of a production cell system (PCS) (taken from [19]) as a working example to provide "precise and unambiguous specifications of the system requirements and design (structure and behavior) and [establish] traceability links between requirements and design" Nejati *et al.* [18].

The purpose of PCS is to transform metal blanks into forged plates (by means of a press). Nejati *et al.* [18] focus on interactions between two devices of the cell: the feed belt and the (rotary) table.

> The cell operator puts the blanks one by one on the feed belt and the belt conveys them to the table. The table then rotates and lifts to put the blanks in the position where a robot arm can take them. [18]

Figure 6 shows a picture of the feed belt, table, and robot arms.

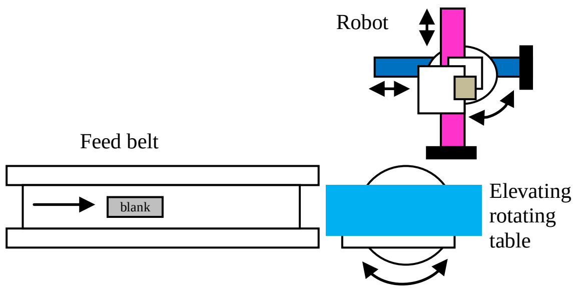

Figure 6. Fragment of the production cell system (partial, modified description from [18], referencing [19])

Nejati *et al.* [18] start with Phase I - Requirements Specification, which includes developing a system context diagram to represent the context with SysML blocks representing the context and SysML parametric diagrams to describe the environmental assumptions (see Figure 7). Then a System-level requirements diagram and a Use case diagram are developed. "Use cases represent the functionality of the software part of a system from an external point of view" [18]. The structure of the system is described using SysML block definition diagrams and internal block diagrams. Figure 8 shows a partial view of some of the diagrams involved.

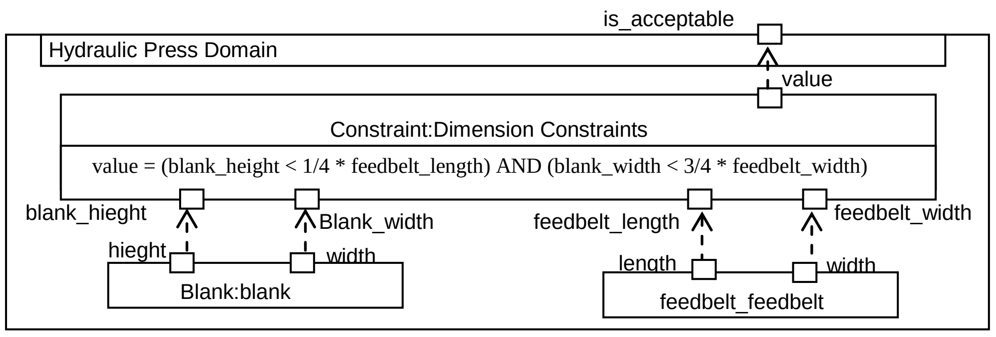

Figure 7. SysML parametric diagram expressing an assumption (from [18])

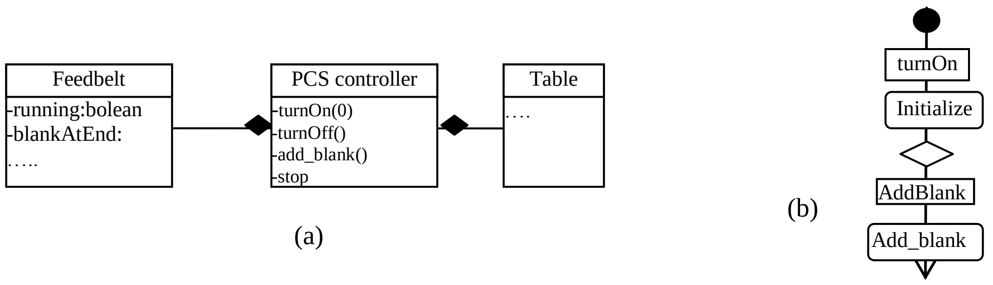

Figure 8. A fragment of design diagrams for PCS: (a) block definition diagram, and (b) activity diagram (partial, from [18]).

In scrutinizing this approach, SysML is supposed to produce a holistic representation of the problem domain. According to Finance [20],

> SysML makes it possible to generate specifications in a single language for heterogeneous teams, dealing with the realisation of the system hardware and software blocks. Knowledge is thereby captured through models stored in a single repository, enhancing communication throughout all teams. In the long term, blocks can be reused as their specifications and models enable suitability assessment for subsequent projects.

Nevertheless, in spite of the compactness of SysML in comparison with UML, problems remain, related to multiplicity of diagrams and lack of a holistic diagrammatic view of the system. "System" here is not limited to an IT system but covers the generic meaning embedded in this term. This multiplicity of diagrams is probably based on the idea of managing complexity by dividing the problem, but this principle can be applied only if you have a unified description of the problem. In the case under discussion, the only underlying complete specification is the textual requirement (narrative use case) given in natural language. Imagine implementing a construction project using different types of diagrams, e.g., space divisions, electrical connections, communication lines, etc., all based on a textual description instead of the architectural drawing.

When the PCS project is discussed with non-SysML oriented stakeholders, SysML use cases and sequence diagrams are probably used to explain the design and how it works; otherwise, most features of SysML diagrams look like gibberish to these stakeholders. There is no "blueprint map" that all participants can utilize as a communication tool, as in the case, say, of construction of a high-rise, where an architectural drawing is at the center of communications for all involved parties.

5. FM Application to Control of a Production Cell

Figure 9 shows the FM representation corresponding to Nejati *et al.*'s [18] fragment of a production cell system, modeled according to our best understanding of the presented description. The purpose of developing this diagram is to compare the two representations side by side, allowing the reader to validate the claim that FM provides a holistic conceptual map of the system discussed above.

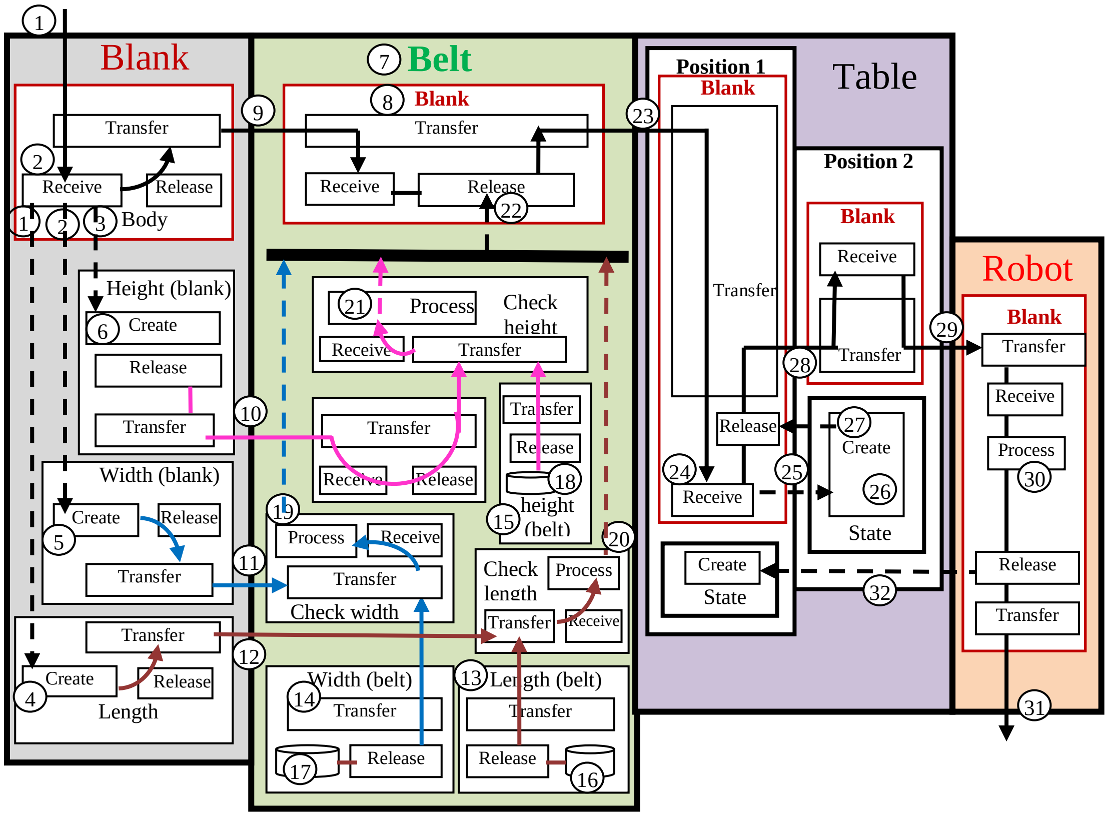

Figure 9. FM representation of the fragment of a production cell system

Figure 9 includes four main spheres: Blank, Belt, Table, and Robot; each includes several subspheres. First, blanks are fed to PCS (circle 1), and the arrival of each triggers (2, 3, and 4) creation of information on Length, Width, and Height (5, 6, and

7), respectively. “Creation” here means the presence of this information in the Blank sphere. How this information enters the system is not given by Nejati *et al.* [18]. The information could be input through a screen to the system, or in the mere process of feeding a Blank, when information could be generated by sensors. In either of these cases, an FM description can be supplemented to trigger this “creation” of information.

“Feeding the blank to PCS” causes emergence of the Blank sphere with all of its subspheres: Physical blank, Length, Width, and Height information. This appearance of the Blank sphere is the first scene in the system and immediately causes the appearance of the Belt sphere (7), which includes a Blank subsphere (8). Note that these spheres are conceptual contexts, not physical. Blank and Belt spheres are analogous to a split screen in a movie where one half shows only the Blank while the other shows the Belt including the blank. Actually, both scenes coexist physically but they can be conceptually different, *e.g.*, the first camera shows aspects (*e.g.*, roughness) of the Blank that do not appear in the second scene. Of course the second camera has a wider field of view than the first.

Also, the ordering of the two spheres (Blank and Belt) appears (conceptually) in terms of the flow of the Blank to the Belt (9). If it were a movie, the Blank would appear (1) in the conceptual representation before the appearance of the belt, and placing it on the Belt is part of this flow. In addition, the information about Length, Width, and Height flows (10, 11, and 12) to the Belt sphere to be checked. This checking uses the Belt measurement of Length, Width, and Height (13, 14, and 15). We assume that these values are stored in the Belt subspheres (16, 17, and 18). Comparing the values of Length, Width, and Height of Blank with the corresponding stored values of the Belt triggers (19, 20, and 21) permitting or not permitting the Blank to be released to the Table sphere (22). The wide joint bar is used to indicate that triggering depends on the results of all three comparisons. FM can be supplemented with all types of notions from other modes, including synchronization and logical symbols that are overlaid over the basic FM representation.

There is no indication in Nejati *et al.*’s [18] description about what to do in case of not satisfying the measurements constraints. In FM, we can insert an “if statement” in the Release stage that diverts the Blank to, say, a junk sphere.

Assuming the Blank satisfies the measurement constraints, it flows to Position 1 subsphere of the Table sphere (23). All types of constraints can be embedded in the Transfer stage. When the Blank is received in Position 1 (24), its arrival triggers (25) the creation of Position 2 (26). In FM representation, it is possible to add the method of this changing of position, such as activation of a part of the table (subsphere) to create a movement that results in Position 2. Notice that a *state* and a *movement* are flowthings that can be created, processed, *etc.*

The creation of Position 2 triggers (27) releasing the Blank to flow to that subsphere (28). This is a conceptual flow and it means the appearance of Position 2 with the Blank as the current state of the Table. Position 2 concurrently with Blank indicates that the Blank flows to the Robot (29), i.e., the Blank enters the context of the Robot. Again, this is a conceptual flow in which two spheres (Position 2 and Robot) share the same physical space (see Figure 10).

The Robot processes the Blank (30), then the Blank flows outside the system (31; *e.g.*, is taken out), and the Table is triggered (32) to return to Position 1 in preparation for the arrival of the next Blank.

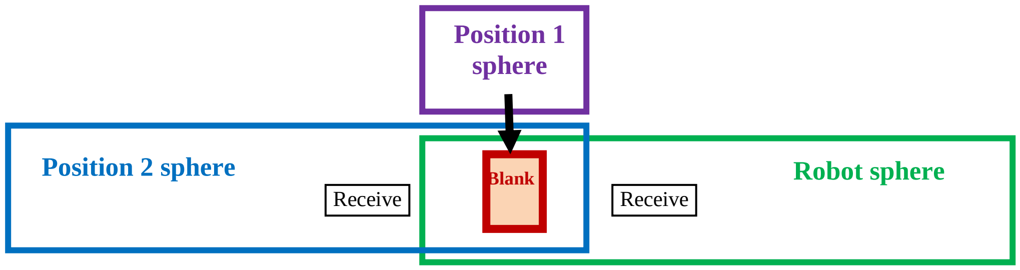

Figure 10. The flow of the Blank from Position 1 to Position 2 implies its flow to the Robot sphere (Robot can "see" the Blank)

6. Conclusion

This paper proposes a new diagrammatic representation method that can serve as an underlying framework for a conceptual schema that reflects the realities of an organization and its operations. The framework can be integrated into the UML system of 13 types of diagrams; however, it is not clear at this point how to accomplish this. It is clear that the FM methodology is a move toward an integrated approach to the problem of representation in this type of application.

The FM representation itself furnishes only a foundation that can be supplemented by tools from other models such as logical operators (e.g., AND), synchronization notions (Petri nets join), constraints specifications, sequential and parallelism control, and use case narratives. These can be superimposed on the basic FM diagram.

References

[1] J. Evermann and Y. Wand, "In Proceedings of the 20th International Conference on Conceptual Modeling, Edited H. Kunii, S. Jajodia, and A. Solvberg, **(2001),** http://www.mcs.vuw.ac.nz/~jevermann/EvermannWandER01.pdf.
[2] C. Kobryn, Comm. ACM, vol. 45, no. 1, **(2002),** pp. 107-110.
[3] D. Dori, Comm. ACM, vol. 45, no. 1, **(2002)**, pp. 82-85.
[4] D. Dori, J. Database Manage, vol. 12, no. 1, **(2001)**, pp. 4-14.
[5] P. Shoval and J. Kabeli, J. Database Manage, vol. 12, no. 1, **(2001),** pp. 15-25.
[6] V. Hoffmann, H. Lichter, A. Nyßen and A. Walter, J. Object Tech, vol. 8, no. 3, **(2009).**
[7] M. Wang, "Issues Inform. Syst", vol. 9, no. 2, **(2008**), pp. 538-543.
[8] P. Kruchten, "The Rational Unified Process: An Introduction (3rd Ed.)", **(2004),** ISBN 0-321-19770-4.
[9] S. Al-Fedaghi, "Int. J. Database Theory Appl", vol. 6, no.3, **(2013**), pp. 59-70.
[10] S. Al-Fedaghi, "J. Next Gener. Inform", Tech., vol. 4, no. 3, **(2013**), pp. 30-38.
[11] S. Al-Fedaghi, "Int. J. Softw. Eng. Appl", vol. 7, no. 2, **(2013**), pp. 171-182.
[12] S. Al-Fedaghi, "Conceptualization of business processes", IEEE Asia-Pacific Services Computing Conference (IEEE APSCC 2009), Biopolis, Singapore, **(2009)** December 7–11**.**
[13] J. Holt and S. Perry, "SysML for Systems Engineering, Institute of Engineering and Technology, **(2008).**
[14] F. O. Hansen, SysML: A modeling language for systems engineering [slides], **(2010),** http://staff.iha.dk/foh/Foredrag/SysML-SystemEngineering-DSFD-15-03-2010.pdf.
[15] S. Friedenthal, A. Moore and R. Steiner, "A Practical Guide to SysML: The Systems Modeling Language", Elsevier, **(2011),** ISBN 0123852064, 9780123852069.
[16] T. Weilkiens, "Systems Engineering with SysML/UML: Modeling", Analysis, Design. Elsevier, **(2007)**
[17] OMG Systems Modeling Language, The Official OMG SysML site. Accessed September, (**2013**), http://www.omgsysml.org/.
[18] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand and T. Coq, "Inform. Softw", Tech., vol. 54, no. 6, **(2012),** pp. 569-590.

[19] C. Lewerentz and T. Lindner, Editors, “Formal Development of Reactive Systems: Case Study Production Cell, vol. 891 of LNCS, Springer, **(1995).**
[20] Guillaume Finance, SysML Modelling Language explained, **(2010),** http://www.omgsysml.org/SysML_Modelling_Language_explained-finance.pdf.