BPMN Requirements Specification as Narrative

Sabah Al-Fedaghi
Computer Engineering Department
Kuwait University
Kuwait
sabah@alfedaghi.com

***Abstract*—The first two phases of the software development process include a requirements analysis stage that demands conceptualization of a "real world domain" and the design stage of the software product. UML-based diagrams are typically used to model systems and make them readable. In this paper we view conceptualization of a piece of reality related to a software system as analogous to a narrative or script created to describe a sequence of events. As an application area, we concentrate on activity diagrams used in BPMN. Examination of typical BPMN representation shows that the resultant picture is fragmented into conceptual gaps and discontinuities. Based on such a perspective, the focus is on maintaining continuity across parts and along the production process of software. To preserve continuity, we propose using the notion of flow as an initial foundation for the conceptualization process.**

Keywords-Activity diagram, BPMN, UML, conceptual model, narrative

I. INTRODUCTION AND MOTIVATION

An information system (IS) should reflect some part of reality and its events. Consequently, building an IS begins by determining requirements as part of a real-world domain. The resulting conceptual picture serves as a guide for the subsequent information system design phase, including a description of the software system under development. According to Peylo [6],

> Requirements engineering is a central part of software projects. It is assumed that two thirds of all errors in software projects are caused by forgotten requirements or mutual misunderstandings in the requirement gathering process. Due to the inherent structure of project planning and the project management process, it is very unlikely that this problem will be solved unless the process itself is changed or we develop tools that possess some intelligence to facilitate the assessment of requirements

Object-oriented methods and languages (e.g., UML) are typically used to describe a software system. Researchers have examined and proposed extending the use of object-oriented software design languages such as UML to apply them at the conceptual level (e.g., [7]). According to Evermann [6], "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."

In this paper, we concentrate on a specific UML structure, activity diagrams, as applied as a conceptualization tool in BPMN. UML activity diagrams are described as the "flow charts" of object-oriented methodology. The problem with extending object-oriented models and languages is "that such languages [e.g., UML] possess no real-world business or organizational meaning; i.e., it is unclear what the constructs of such languages mean in terms of the business" [6]. The object-oriented IS design domain deals with objects and attributes, while the real-world domain deals with things and properties. According to Storrle and Hausmann [9], in UML, "activity diagrams have always been poorly integrated, lacked expressiveness, and did not have an adequate semantics in UML." With the development of UML 2.0, "several new concepts and notations have been introduced, e.g., exceptions, collection values, streams, loops, and so on" [9].

This paper proposes an alternative approach to specify system requirements. The approach analyzes the relationship between two types of conceptualizations—technical conceptualization and artistic conceptualization—for the purpose of focusing on a main feature of conceptualization: *continuity*.

II. CONCEPTUALIZATION

We view conceptualization as of two types: functional and artistic. *Functional conceptualization* is used for the purpose of representing a piece of reality to be used in building an information system. The resulting artifacts are meant to represent functional requirements. Take for example a UML *use case*, which describes an interaction as a sequence of single steps and events to achieve a specific goal. In this context, there are several representation schemes.

> The meaning (or semantics) of the use case is not represented by the well defined building blocks of the formalism …, but shall constitute itself (helped by various annotations) in the mind of the reader. This approach is quite common but prone to misunderstandings. [6]

Furthermore, Peylo [6] states that

> Due to their seeming clarity and formality they are often over-estimated. Nevertheless, they are deceptive with respect to their precision and expressiveness. Their main limitations are:
> 1. Weak and not well defined semantics of relations.
> 2. The expressiveness of graphical representation schemes is limited per se to a fragment of first order logic
> 3. Generally, it is not possible to decide by the study of a use case whether the process flow may lead to the desired result (i.e. the system output may be achieved, given the set of input).

Artistic conceptualization is also generated for the purpose of representing a part of reality. It can be exemplified by narratives, scripts of movies, and comic books.

Both types of conceptualization are strongly founded on language. Their orientations are different, as shown in Fig. 1. Artistic conceptualization captures reality but seeks to release its content in an expanded universe of meanings and interpretations. Functional conceptualization seeks precision in releasing its content by narrowing its meaning and interpretation. An important aspect of both types of conceptualization is *continuity,* as described in the next section.

A notion related to artistic conceptualization is that of "operation concept," which includes concept analysis. Concept analysis is an overall "system development process" for analyzing an operational environment and characterizes a proposed system from the user's perspective.

> [An operation concept] document should, in contrast to a requirements specification, be written in narrative prose, using the language and terminology of the users' application domain. It should be organized so as to tell a story, and should make use of visual forms (diagrams, illustrations, graphs, etc.) whenever possible [5].

However, this does not focus on the notion of *flow* (a fundamental concept in our approach that will described later) even through it recommends "scenarios [that] are specified by recording, in a step-by-step manner, the sequences of actions and interactions between a user and the system" [5].

III. CONTINUITY

In a system, continuity indicates uninterrupted connection and succession. In the production of film and television, a script supervisor is concerned with maintaining *continuity* across shots and along the production process. In comic books, *continuity* means contiguous events "in the same universe."

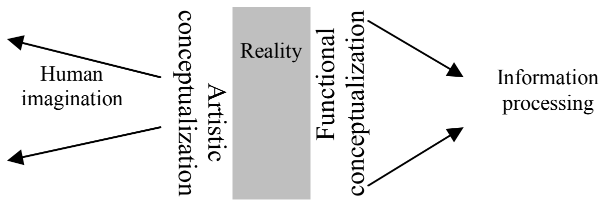

Figure 1. Orientations of artistic and functional conceptualizations.

In business, the notion of continuity/security arises when planning for permanence of critical business processes in case of security failure. It is a notion related to survivability, load balancing, and redundancy.

In beginning mathematics a function is continuous if we can draw its graph without taking the pencil off the page. A *discontinuity* is a point where a function is not continuous.

Ivic [4] defines *discontinuity* as "the lack of … logical sequence." According to Webster's New World College Dictionary [10], discontinuity means "a lack of continuity or *logical sequence*, or a gap or a break… this could mean a break in the chronological sequence, or a very fragmented structure in poetry." Discontinuity is an undesirable feature in literature. "Discontinuity in a novel interrupts the *flow* of the story, ..." [10, italics added].

In architectural design, continuity is "the measurement of the *completeness* of the sidewalk system with avoidance of *gaps*… the pedestrian sidewalk appears as a single entity within a major activity area or public open space" [11, italics added].

In a software system, *discontinuity* may be a positive feature for security. "System discontinuity emphasizes security over compatibility by removing those constructs in our system software which lead to security holes in applications" [12]. Such strategy removes parts of the interfaces "both of programming languages and operating systems which have proven to engender the greatest number of security holes." Such a proposal assumes completeness. In analogy, to secure a physical territory, subterritories can be disconnected; however, the interior of each piece of territory should be completely known (e.g., surveyed).

A conceptualization of reality needs a type of continuity: *logically sequential progression*. This can be thought of as reflecting the Aristotelian notion of organic unity, where each component of a task is a necessary part of a whole.

Continuity is a necessary feature for designers. After producing a conceptual representation, designers will seek connections through temporal continuity, causality, or some commonality such as presence in the same sphere.

In general, the notion of continuity is a phenomenon that involves a gradual transition without abrupt changes or discontinuities. We view it as the property of connectedness of conceptual space of events. When a conceptualization seems fragmentary, we look at the represented world. Is there an underlying represented "reality" that can be pieced together? Are there missing entities or connections? Are there discontinuities between spheres? Do conceptual parts have gaps that can't be assertively filled?

We show the importance of continuity through scrutinizing BPMN activity diagramming. To provide opportunities to contrast continuous and discontinuous representations, we next review a flow-based conceptualization that can be used for modeling activities.

IV. FLOWTHING MODEL (FM)

A flow model is a uniform method for representing things that "flow," i.e., things that are exchanged, processed, created, transferred, and communicated [1, 2]. "Things that flow", called flowthings, include information, materials (e.g., in manufacturing), and money. To simplify this review of FM, we introduce the model in terms of information flow. Information occurs in five states: transferred, received, processed, created, and released, as illustrated in Fig. 2. Here, we view a "state of information" in the sense of properties; for example, water occurs in nature in the states of liquid, solid, and gas.

Fig. 2 also represents a transition graph, called a flowsystem, with five information states and arrows representing flows among these states. Information can also be stored, copied, destroyed, used, etc., but these are secondary states of information in any of the five generic states. In Fig. 2, flows are denoted by solid arrows. Flows may *trigger* other types of flow, denoted by dashed arrows, as will be discussed.

The environment in which information exists is called its sphere (e.g., computer, human mind, organization information system, department information system). The flowsystem is reusable because a copy of it is assigned to each entity (e.g., software system, vendor, and user). An entity may have multiple flowsystems, each with its own flowsystem. It is possible to have flowsystems of different flowthings: requests, invoices, plans, and actions. These are, like information, flowthings that can be received, processed, created, released, and transferred.

A flowsystem may not necessarily include all states, for example, conceptualization of a physical airport can model the flow of passengers: arriving (received), processed (e.g., passports examined), released (waiting to board), and transferred (to planes); however, airports do not create passengers (ignoring the possibility of an emergency where a baby is born in the airport). In this case, the flowsystem of the airport includes only passenger states of received (arrival), processed (e.g., passports), released (waiting for boarding), and transferred (on the plane).

As we mentioned previously, we view a system as the environment in which information exists, called its sphere. A system is also viewed as a complex of flowsystems.

The states shown in Fig. 2 are exclusive in the sense that if information is in one state, it is not in any of the other four states. Consider a piece of information x in the possession of a hospital. Then, x is in the possession of the hospital and can be in only one of the following states:

1. x has just been collected (*received*) from some source, e.g., patient, friend, or agency, and stored in the hospital record waiting to be used. It is received (row) information that has not been processed by the hospital.
2. x has been *processed* in some way, converted to another form (e.g., digital), translated, compressed, etc. In addition, it may be stored in the hospital information system as processed data waiting for some use.
3. x has actually been *created* in the hospital as the result of doctors' diagnoses, lab tests, produced by processing current information (e.g., data mining), and so forth. Thus, x is in the possession of the hospital as created data to be used.

If a piece of information is copied, then the new piece of information is a different instance of a flowthing (e.g., one is stored, and one is transferred).

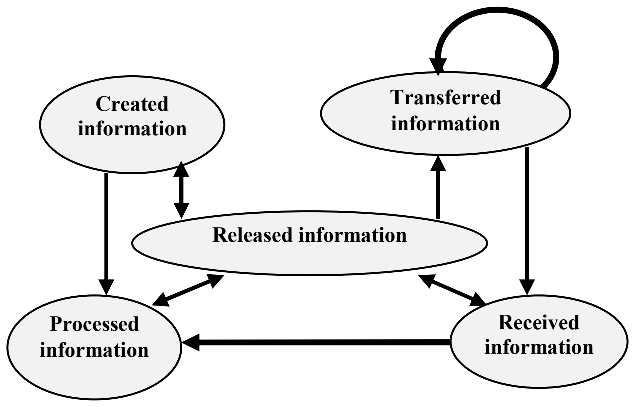

Figure 2. State transition diagram of FM with possible triggering mechanism.

4. x is being released from the hospital information sphere. It is designated as released information ready for transfer (e.g., sent via DHL). In an analogy of a factory environment, x would represent materials designated as ready to ship outside the factory. They may actually be stored for some period waiting to be transported; nevertheless, their designation as "for export" keeps them in such a state.

5. x is in a transferred state, i.e., it is being transferred between two information spheres. It has left the released state and will enter the received state, where it will become received information in the new information sphere.

It is not possible for processed information to directly become received information in the same flowsystem. Processed information can become received information in another flowsystem by first becoming released information, then transferred information, in order to arrive at (be received by) another flowsystem.

Consider the seller and buyer information spheres shown in Fig. 3. Each contains two flowsystems: one for the flow of orders, and the other for the flow of invoices. In the seller's infosphere, processing of an order triggers (circle 3) the creation of an invoice in the seller's information sphere, thus initiating the flow of invoices.

The reflexive arrow of the transfer state shown in Fig. 2 (above) denotes flow from the transfer state of one flowsystem to the transfer state of another.

In Fig. 3, the Buyer creates an Order that flows by being released and is then transferred to the Seller. The "transfer components" of the Buyer and the Seller can be viewed as their transmission subsystems, while the arrow between them represents the actual transmission channel.

V. BPMN ACTIVITY DIAGRAM

Business Process Modeling Notation (BPMN) is popular in some communities of practice and "in some cases may be locally mandated" [8]. Therefore, it is useful to utilize it as an area where different activity conceptualizations are compared.

When developing a business system, it is essential to first produce a general conceptual description of *activities*. The activities pose scenarios that represent the circumstances of events. The resultant description is a model of overall activities, subactivities, and connections among them. This conceptualization liberates designers to produce neutral specifications not oriented to any actual current methodology of conducting business. It also represents a common understanding of system operations shared by technical and nontechnical individuals involved in the project.

BPMN shows activities within swimlanes, which represent different performers as nodes in the Business Node Connection Model. Fig. 4 illustrates the basic form of a BPMN diagram, in the context of a travel planning activity [8 - Citizant Corp.]. Fig. 5 shows the corresponding FM representation. According to Sowell [8],

> This all-in-one notation can be very helpful and time-saving when the architecture in question is an As-Is architecture, because *all the relevant information* is known, and merely needs to be captured. [Italics added]

We claim that the activity diagram shown in Fig. 4 exhibits a fragmented conceptualization of reality. The workflow description items form a narrative that is created to describe a sequence of events. Consequently, we go one item at a time, as follows. We assume that the software designer is the reader of such a narrative.

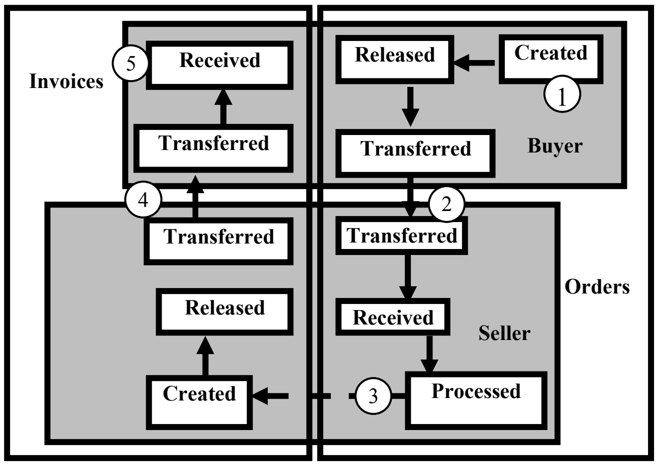

Figure 3. Order flow triggers invoice flow.

Consider the following scenario in Fig. 4.

Travel agent: Research Travel Options
Traveler: Select Itinerary
Travel agent: Make Reservation
Traveler: Submit Payment
Travel agent: Confirm reservation
Traveler: Verify Itinerary

The arrows in the figure seem to indicate control flow. The semantics involved are as follows:

The travel agent researches travel options,
the traveler selects an itinerary,
the travel agent makes the reservations,
the traveler submits payment,
the travel agent confirms the reservation,
the traveler verifies the itinerary.

Here we see a discontinuity. For example, in the sequence: [the travel agent confirms the reservation →
the traveler verifies the itinerary] the events seem to jump.

A corresponding scenario with continuity would be as follows:

The travel agent researches travel options,
the search by the travel agent produces a list of options,
the travel agent sends the list to the traveler,
the traveler selects an itinerary from the list,

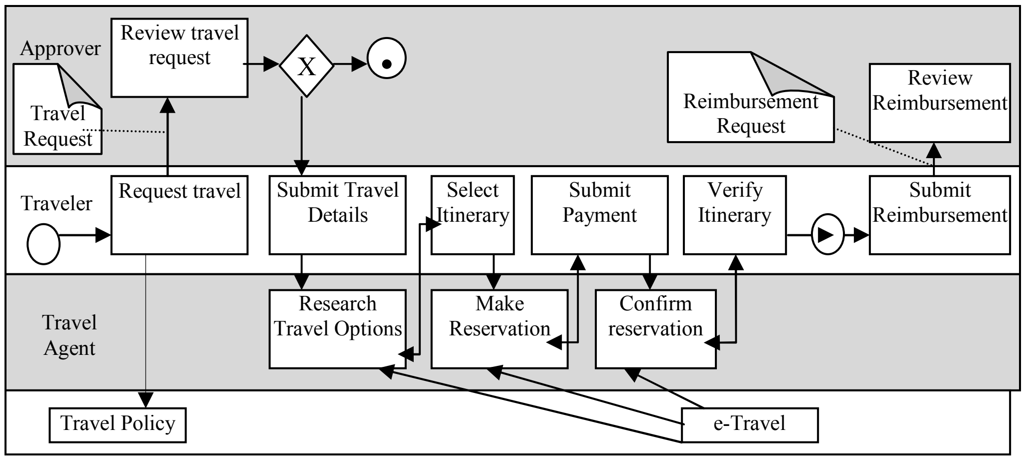

Figure 4. A Simple BPMN Diagram [8].

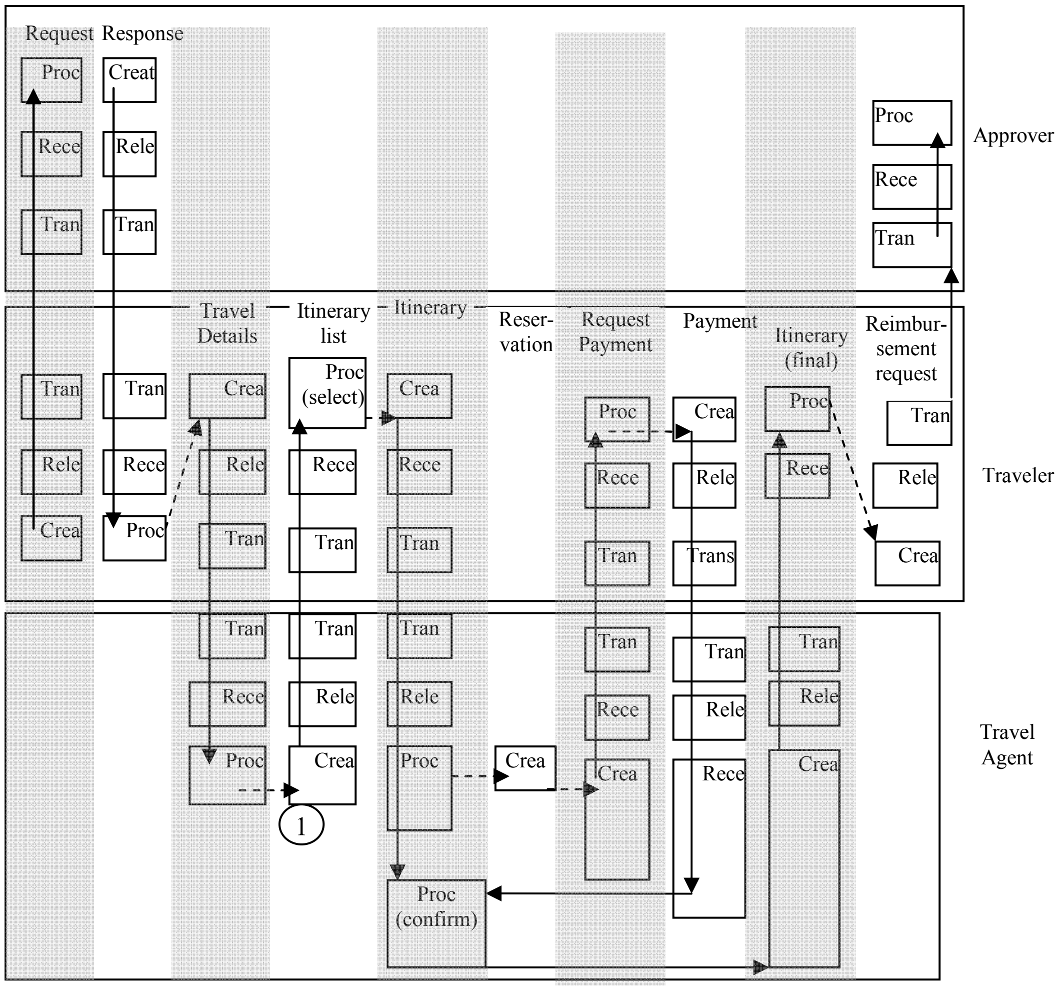

Figure 5. Flow-based representation of a Simple BPMN Diagram

the traveler sends the itinerary to the travel agent,
the travel agent makes the reservation,
the travel agent issues a payment invoice,
the travel agent sends the invoice to the traveler,
the traveler receives the invoice,
the traveler makes payment(e.g., money order)
the traveler sends payment to the travel agent
the travel agent receives payment from the traveler,
the travel agent confirms the reservation,
the travel agent sends the itinerary to the traveler,
the travel agent confirms the reservation, then
the traveler verifies the reservation.

Fig. 5 reflects such continuity. Starting at circle 1, the travel agent creates an itinerary that flows to the traveler, which triggers him/her to select a single option (itinerary) that flows back to the travel agent, who processes it and (1) makes a reservation, and (2) creates an invoice. The invoice is sent to the traveler, who makes (creates) payment, which arrives at the travel agent. The travel agent confirms the reservation, and sends the final itinerary to the traveler. Upon receiving the itinerary, the traveler processes it to verify it.

This flow-based description is similar to a comic book, where a stream of events flows in a continuous fashion. Flowthings such as requests, lists, and invoices flow like a ping pong ball between players.

VI. WITH WORKFLOW DESCRIPTION

"Use case" as a modeling tool provides a software-independent description of the processes to be automated.

> The IT team must have descriptions of the business that allow team members to make informed decisions, including an unambiguous specification of the business process that details relevant value and cost factors. Business use cases are documented via specifications that consist of both textual workflow descriptions and one or more Unified Modeling Language (UML) activity diagrams. [3]

Consider Fig. 6, which provides an example of a business use case specification [3]. The activity diagram provides a pictorial representation of the workflow structure described in the following business use case text. [3] gives the corresponding workflow description for Fig. 6. For lack of space, we discuss the first three steps as follows:

• The Customer Sales Interface initializes contact.

• If the Customer Sales Interface determines that initial opportunity work is complete, then the Customer Sales Interface sends a proposal request to the Proposal Owner.

• Otherwise the Customer Sales Interface searches for alternatives. [3]

As stated previously, the workflow description items form a narrative that describes a sequence of events. We assume that the software designer is the reader of such a narrative.

• The Customer Sales Interface (CSI) initializes contact.

From such a description, implicitly (from the name), we understand that there is a customer. Contact denotes communication, thus, it seems that the designer would understand that CSI creates something (e.g., a message) and then executes the contact. To maintain continuity and completeness in the initial step, we must explicitly state that something is created, as follows.

CSI creates an offer and communicates it to customer.

Here we ignore the issue of what type of information is involved in such a creation.

• If the Customer Sales Interface determines that initial opportunity work is complete, then the Customer Sales Interface sends a proposal request to the Proposal Owner (PO).

This scenario includes missing pieces. How does the designer understand that the "determination" is the result of receiving some type of communication from the customer? It is possible that the designer thinks that embedding some type of information about communication with a customer is unnecessary since the determination is based on informal contact. It is highly improbable that contact with a customer is non-recorded informal contact. We can rewrite this as follows.

CSI receives a response from the customer, processes the response, then If CSI determines that initial opportunity work is complete, CSI sends a proposal request to the PO.

The whole process can be described as flows of offers, responses, and requests as partially shown in Fig. 7.

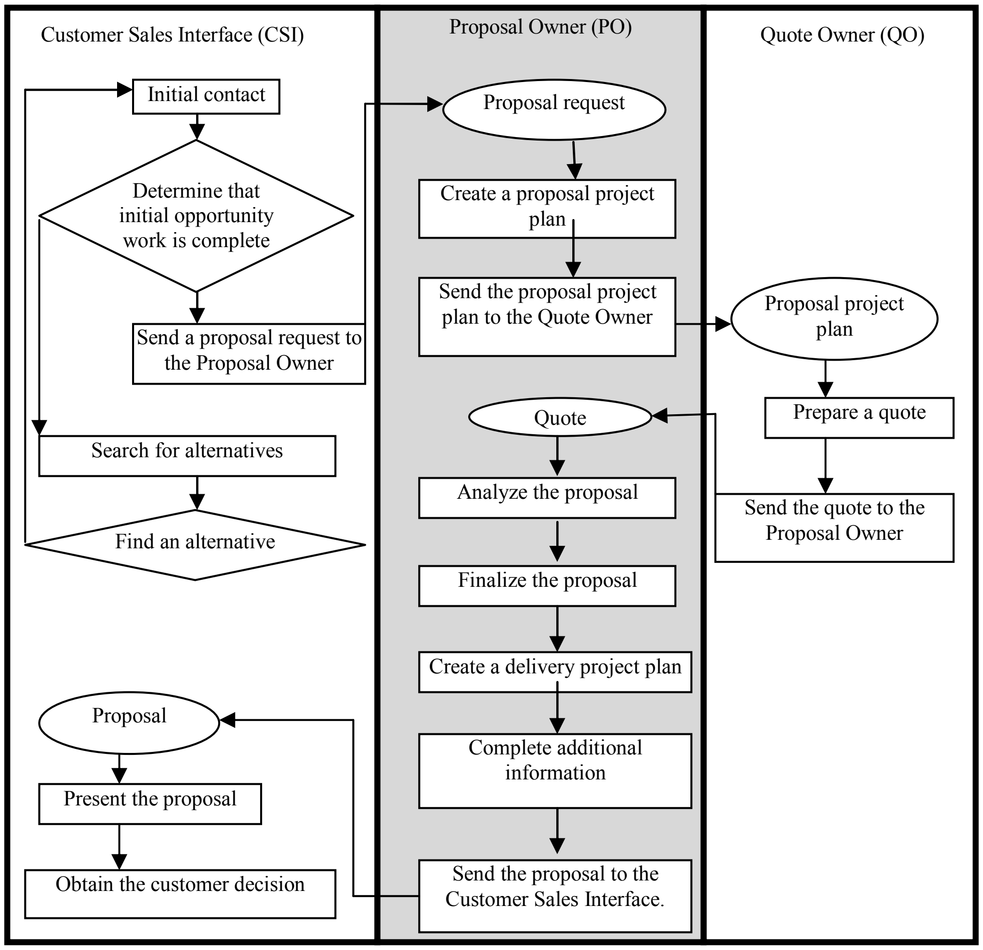
Customer Sales Interface (CSI)
Proposal Owner (PO)
Quote Owner (QO)
Initial contact
Determine that initial opportunity work is complete
Send a proposal request to the Proposal Owner
Search for alternatives
Find an alternative
Proposal
Present the proposal
Obtain the customer decision
Proposal request
Create a proposal project plan
Send the proposal project plan to the Quote Owner
Quote
Analyze the proposal
Finalize the proposal
Create a delivery project plan
Complete additional information
Send the proposal to the Customer Sales Interface.
Proposal project plan
Prepare a quote
Send the quote to the Proposal Owner

Figure 6. Example of a business use case specification (From [3]).

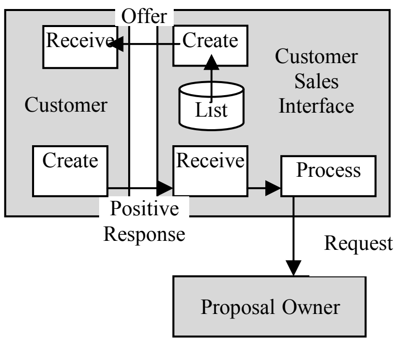

Figure 7. Non-programming conceptualization of part of the example.

Notice that a general conceptualization (shown partially for lack of space) in Fig. 7 reflects a "forest-level" of flows in the piece of reality being abstracted. There is the flow originating from CSI to customer, and another flow originating from customer that may reach PO. The *need* for flow has been expressed previously in a discussion of Peylo's [6] "flow of action" in scripts, and "flow of the story" in [11].

Notice also the general level of conceptual mapping in FM. When designing a city, the designer does not specify at intersections that green means go and red means stop. These details (types of processes in FM) come at a lower level of abstraction. Thus, it is not necessary, in our example, to specify at this level, that "If the Customer Sales Interface determines that initial opportunity work is complete, then the Customer Sales Interface sends a proposal request to the Proposal Owner." It is sufficient to indicate at this point that:

- Responses are received from customers, processed, and according to this process a request is sent to Proposal Owner.

FM description draws a conceptual topology of flows of data, leaving the interior of the process (e.g., specification of decision criteria) to a later stage. Accordingly, the designer can visualize the total procedure as:

For list of customers
Send an offer
Receive a response
Process the response
According to the results of processing, send/do not send a request for Proposal Owner

Note that there is no "if" statement in this procedure, because "if" triggers specification of the criteria for a decision.

Additionally, going back to the narrative of workflow of [3], we see the following.

• *The Quote Owner (QO) prepares a quote.*
• *The Quote Owner sends the quote to the Proposal Owner.*

Here one wonders why "prepare" is used, instead of "create," as used previously by [3]. "Create" is more suitable because it is a flow-oriented term: QO creates (originates) quotes that flow to PO.

In Fig. 6, there are odd arrows (dataflow? control flow?) from a process to an object, such as the arrow from "Send the proposal project plan" to the "Quote Owner", and the arrow from "Send the quote to the Proposal Owner" to "a quote".

We stop here reviewing the rest of the workflow and activity diagram because it is clear at this point that such a description is "narrative-wise", is a fragmented conceptualization that is filled with gaps, and discontinuities.

Finally, we note the uncontrollable use of many verbs: "initializes", "determines", "searches", "finds", "sends", "prepares", "creates", "analyzes, "finalizes", "completes", "presents", and "obtains". This style of specifying flow among processes is a frail feature in any good "conceptual narrative". In contrast, FM uses only five flow-oriented operations: receive, process, create, release, and transfer.

Clearly, we are not introducing a completely new methodology for specifying requirements; rather we describe a general approach that emphasizes flow and continuity of requirements description.

VII. CONCLUSION

This paper introduces the concept that a piece of reality related to a software system can be conceptualized analogous to a narrative or script created to describe a sequence of events. This is demonstrated by applying it to activity diagrams used in BPMN utilizing flow-based model. The resultant description maintains continuity across parts and along the production process of software. Further research would explore applying the concept to other software diagramming tools.

REFERENCES

[1] S. Al-Fedaghi, "Conceptualization of business processes," IEEE Asia-Pacific Services Computing Conference (IEEE APSCC 2009), Dec 7-11, 2009, Biopolis, Singapore.

[2] S. Fedaghi, "Scrutinizing UML activity diagrams," 17th International Conference on Information Systems Development (ISD2008), Paphos, Cyprus, August 25-27, pp. 59-67, 2008.

[3] A. Frankl, "Validated requirements from business use cases and the Rational Unified Process," IBM, 15 Aug 2007, accessed June, 2010. http://www.modernanalyst.com/Resources/Articles/tabid/115/articleType/ArticleView/articleId/52/Validated-requirements-from-business-use-cases-and-the-Rational-Unified-Process.aspx

[4] C. Ivic, "Review of discontinuities: new essays on Renaissance literature and criticism," Early Modern Literary Studies 5, 2, September, 1999, 8.1-6, accessed June 2010. http://purl.oclc.org/emls/05-2/ivicrev.htm

[5] R. E. Fairley, R. H. Thayer, and P. Bjorke, "The concept of operations: the bridge from operational requirements to technical specifications," Proceedings IEEE International Conference on Requirements Engineering , 18-21 April 1994, Colorado Springs.

[6] C. Peylo, "On restaurants and requirements: how requirements engineering may be facilitated by scripts," The 4th Workshop on Knowledge Engineering and Software Engineering (KESE 2008), accessed June, 2010. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-425/paper7.pdf

[7] J. Evermann, and Y. Wand, "Towards ontologically based semantics for UML constructs," In: Kunii, H, Jajodia, S, and Solvberg, A (eds.), Proceedings of the 20th International Conference on Conceptual Modeling, pp. 354-367, 2001, Yokohama, Japan.

[8] K. Sowell, "Creating and Presenting Activity Models," SowellEAC Blog, Accessed, December, 2009. http://sowelleac.com/Creating_and_Presenting_Activity_Models.pdf

[9] H. Storrle, and J. H. Hausmann, "Towards a formal semantics of UML 2.0 activities," German Software Engineering Conference, pp. 117-128, 2005. http://wwwcs.uni-paderborn.de/cs/ag-engels/Papers/2005/SE2005-Stoerrle-Hausmann-ActivityDiagrams.pdf.

[10] ENH241, "American Literature before 1860, Discontinuity," Accessed June, 2010. http://enh241.wetpaint.com/

[11] Kansas City, "Walkability Plan. 32. Measuring Walkability," Accessed June, 2010, . http://www.kcmo.org/idc/idcplg?IdcService=GET_FILE&dID=26423&dDocName=019904

[12] J. A. Solworth, "Robustly secure computer systems: a new security paradigm of system discontinuity," Proceedings of the 2007 Workshop on New Security Paradigms, 2007.