International Journal of Control and Automation
Vol.7, No.2 (2014), pp.427-438
http://dx.doi.org/10.14257/ijca.2014.7.2.38

System for a Passenger-Friendly Airport: An Alternative Approach to High-Level Requirements Specification

Sabah Al-Fedaghi

Kuwait University
sabah.alfedaghi@ku.edu.kw

Abstract

Recent advances in large systems (e.g., service-oriented architecture) utilize the notion of choreography, in which there is no central control of the system. Such a task requires first developing a conceptual model to serve as a foundation for specification of choreography-based activities. The European project CHOReOS has built a UML profile for Business Process Model and Notation (BPMN) choreography-modeling capabilities. This paper is concerned with the diagrammatic tools used in the "integrated development process" as exemplified in a passenger-friendly airport model provided to highlight CHOReOS innovation. In this paper an alternative model is proposed as a conceptual base for this development methodology. The UML/BPMN description is reconstructed in terms of the proposed model to facilitate a side-by-side comparison of the two renditions of the same problem. The conclusion is that the proposed alternative approach presents a viable technique for developing a foundation that captures the relevant processes and relationships in systems. In general, the passenger-friendly airport furnishes a sample study case to show that the new diagrammatic method can be applied to high-level requirements specification in any similar development scheme.

Keywords: *conceptual model, high-level requirements specification, diagrams, UML, BPMN*

1. Introduction

With the growth of network-based complex systems, application design has evolved tremendously, as in the case of service-oriented architecture [1, 2], where multiple services *orchestrate* their works. In some cases, called *choreography*, each subsystem interacts independently to accomplish a group goal, with no central control [3]. According to [4], collaboration among multiple services can be approached in terms of a choreography of global collaborative processes and interactions, a behavioral interface of a single service that participates in the choreography, and orchestration of the interactions in which a given service can engage with other services. A number of conceptual characterizations have been described in the literature, such as the European project CHOReOS that aims to investigate the impact of the Future Internet [Ultra Large Scale] on service-oriented systems [5, 6].

> CHOReOS revisits the concept of choreography-centric service-oriented systems to introduce a dynamic development process and associated methods, tools, and middleware. This IDRE (Integrated Development and Runtime Environment) is designed to build robust software systems that implement and coordinate the services in the Ultra Large Scale Future Internet [7].

It targets a set of challenges related to system development, including construction of a conceptual model to serve as the foundation for choreography-based activities. A conceptual model provides a high-level representation that includes relevant entities and relationships among them in a system [5]. It is a picture that describes a real-world domain while excluding technical aspects, to serve as a guide for the subsequent design phase.

According to the CHOReOS Project Team [4], the conceptual model for "the choreography-based Future Internet" uses UML-like notation with textual description, and service choreographies are constructed by extending the existing BPMN 2.0 language [8].

The aspect of CHOReOS of interest in this paper concerns only the tools for choreography modeling based on UML/BPMN to "integrate [the] development process" [9] as exemplified in the "passenger-friendly Airport" model used to highlight CHOReOS innovation. The development method is founded on a "technology-independent characterization of the 'strategy' to be used during the choreography life cycle usual software definition activities, but structured in a CHOReOS-specific ... high-level development process model specified in BPMN2 deployment process model" [9].

> In BPMN 2.0, the focus is not on orchestrations of the activities performed within these participants, but rather on the *external exchange of information (messages) among them*. Thus, a choreography diagram is another way of modeling the "ping-pong game" among different participants without explicitly specifying the internal process flow of each participant…
> Unfortunately, BPMN was defined as a separate metamodel, which provides no interoperability with UML… models that can capture further details on service contracts and messages. [5] (italics added)

Also, according to Silingas [10],

> Model-driven development is a technology that aims to handle software development at a higher abstraction level using models as the main development artifact. However, there is no agreement on how service-oriented systems should be modeled – multiple modeling languages such as BPMN 2.0, UML, … provide fragmented support but fail to cover the challenging aspects of service engineering in the Future Internet.

Scenarios for service choreographies involve complex *workflows* with multiple enterprises [11, 12]. This paper proposes an alternative model, called the Flowthing Model (FM), that is suitable for representing this general notion in CHOReOS since it is fundamentally built on the notion of conceptual *flow*. The example of a passenger-friendly airport in the CHOReOS depiction can be used to show that the proposed flow-based approach can be applied to high-level requirements specification in similar development schemes.

In the next section, one scenario in the passenger-friendly airport, called "Bad weather at destination," is summarized, highlighting and comparing its UML and BPMN diagrams and scrutinizing (a) some of the semantics of its artifacts, and (b) the heterogeneous multilevel diagramming process in which UML and BPMN diagrams are used. Section 3 reviews the flow-based model, called the Flowthing Model (FM), utilized to rediagram the "Bad weather at destination" scenario.

2. Bad Weather at Destination

In the context of CHOReOS, Chatel *et al.* [13] developed requirements specification and scenarios for a passenger-friendly airport using UML-like diagrams:

> The main aim of the use case presented … is to illustrate the usefulness of the CHOReOS software platform for choreography support in the Future Internet, by relying on the airport domain, and its related domain-specific tasks…
> The purpose … is to clarify the individual actions that take place among different actors of the "Passenger-friendly" scenario, as initially described in the DoW. People, services and Things interact with each other throughout the entire sequence of steps that compose the two distinct, but complementary, scenarios that support this specific use case of the CHOReOS project [13].

The definition of the choreographies is based on the BPMN standard, which includes *Flow objects* that "interrupt the sequence flow and may lead to divergence/convergence," and *Connecting object*s that "represent different types of associations between flow objects/data, *etc.*" [13].

This paper focuses on the "Bad weather at destination" scenario because many details are included. The purpose is to contrast the total process of requirements specification in the UML/BPMN-based approach with the new method, FM, proposed in this paper. For the sake of comparison, a substantial description must be summarized from Chatel *et al.* [13]; however, the aim is not to present complete details for understanding, but to show aspects sufficient for judging the features of the two methodologies. The claim is that Chatel *et al.*'s [13] choreographies can utilize FM as a conceptual base that facilitates a holistic style and continuous sequence of events.

2.1. Bad weather scenario

This scenario describes the consequences of rerouting an airplane due to bad weather at its destination airport. Figure 1 shows the steps followed to find an alternative destination airport and keep passengers informed as well as prepared for their unexpected arrival at the new airport.

Examining the figure from a conceptual point of view, one expects it to follow a coherent and systematic specification, to be a methodical and orderly depiction of occurring events while utilizing a few basic notions. For example, consider the arrows in Figure 1. They definitely do not represent a conceptual causal sequence relationship; *e.g.*, "Allocate security facilities" does not necessarily follow or cause "Allocate luggage belt." From a semantic point of view, this mix of relationships creates a great deal of vagueness in knowledge representation and understanding. Avoiding ambiguity such as this is always wise, especially since the application under development is a sensitive application in an important area that deserves precision and consistency.

FM systematically captures the chronology of activities through the notion of flow. *Flow* here refers to the flow of things, similar to flows of electricity, water, and gas shown in the blueprint of a building.

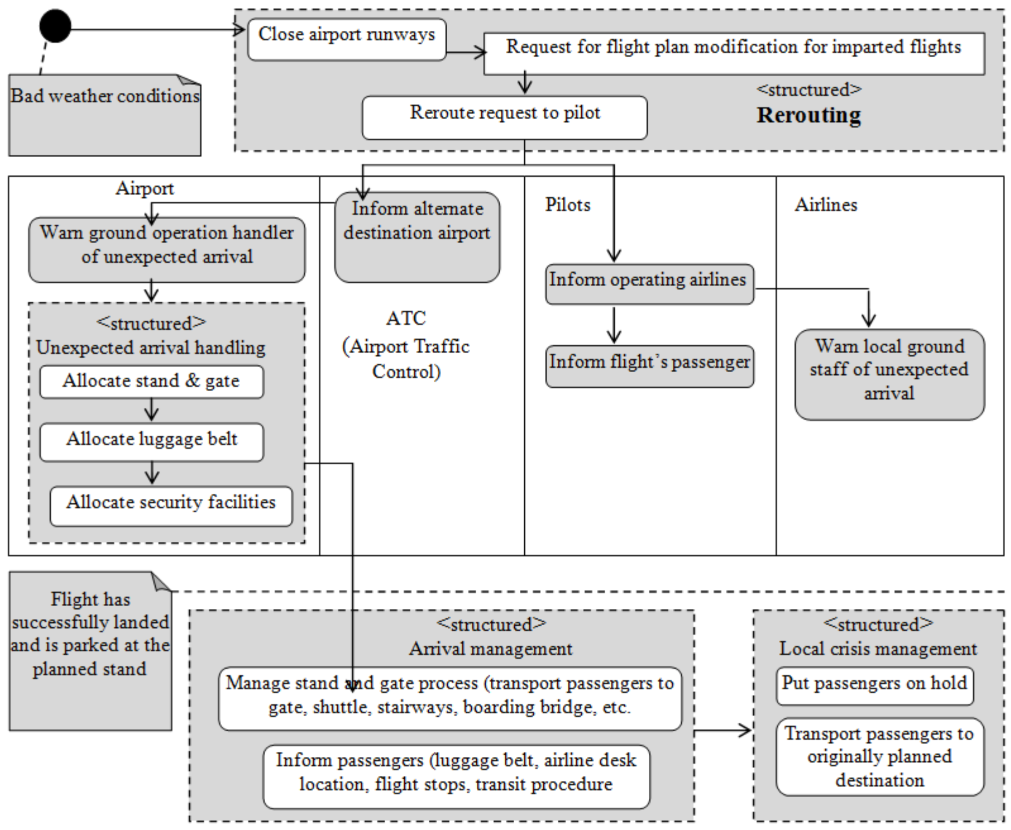

Close airport runways
Request for flight plan modification for imparted flights
Bad weather conditions
Reroute request to pilot
<structured>
Rerouting
Airport
Warn ground operation handler of unexpected arrival
Inform alternate destination airport
Pilots
Airlines
Inform operating airlines
Inform flight's passenger
Warn local ground staff of unexpected arrival
ATC (Airport Traffic Control)
<structured> Unexpected arrival handling
Allocate stand & gate
Allocate luggage belt
Allocate security facilities
Flight has successfully landed and is parked at the planned stand
<structured> Arrival management
Manage stand and gate process (transport passengers to gate, shuttle, stairways, boarding bridge, etc.
Inform passengers (luggage belt, airline desk location, flight stops, transit procedure
<structured> Local crisis management
Put passengers on hold
Transport passengers to originally planned destination

Figure 1. (Partial) UML activity diagram of "Bad weather at destination" (from [13])

Second, the activities mentioned in the figure have no system of categorization. Words such as warn, close, inform, allocate, transport, manage, transport, landed, and parked are used; thus, the model adopts seemingly millions of words that refer to actions in English with no systematic grouping of activities. This causes open-ended representation that is not much different from the ambiguity of natural language texts in which modelers express the same activity in many different terms. In contrast, FM uses six generic actions performed on "things," and things can be subjected to one and only one of these actions at any given time.

Also, in Figure 1, no clear boundaries are shown between original airport and alternative airport. In contrast, FM structure includes hierarchies of spheres (environments) where things move in "flow systems" among spheres and subspheres.

In addition to vague language and structure in the UML diagram of Figure 1, a tabular description (see Table 1) is needed to complement it.

2.2. Step-level requirements

Accordingly, Chatel *et al.* [13] developed step-level requirements with more detailed specifications of the scenario. Figure 2 shows a partial diagram of Rerouting (top box in Figure 1). The diagram seems to serve as a supplementary technical description. The modeling process moves jerkily from the general diagrammatic picture of Figure 1 to the technical depiction in Figure 2.

Table 1. Individual steps in the "Bad weather at destination" scenario (from [13])

Step	Description
Start	An airport is facing bad weather conditions (heavy snow) and reports from weather stations are forecasting poorer weather to come.
1: Rerouting	A decision is made to close all airport runways and flight plans for all scheduled flights need to be modified accordingly (step 2). Airport ATC communicates with pilots for reroute request.
2: Flight plan modification	Flight plan modification Once agreement is found on the final destination between ATC and the pilot: 1. The pilot informs his airline of the flight plan modification and gives new information to his passengers. 2. The alternate destination airport is informed of flight plan modification by ATC.
3:Unexpected arrival warning	The local destination Airport authority warns ground operations handlers (Airport Bus company, Luggage Handling company, Security company, etc.) of the unexpected arrival. The Airline informs warns its own local ground staff (at destination) of the arrival at their airport.
4:Unexpected arrival handling	The local destination Airport authority manages stand and gate, luggage belt and customs and security allocation, in anticipation of unexpected arrival.
5:Arrival management	The local destination Airport authority puts into place stand and gate management (to transport passengers by bus to gate if needed, etc.) and provides information to passengers such as luggage belt location.
6: Local crisis management	The Airline ground staff makes a decision between putting the passengers on hold (for the night) until the weather conditions are restored, or directly transporting the passengers to the originally planned destination (using new flight, bus and/or rail).

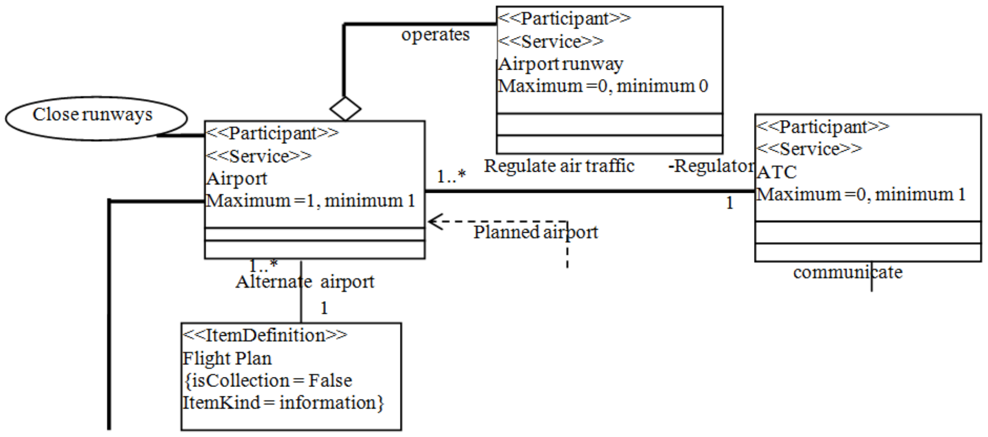


Figure 2. Partial diagram of Rerouting (from [13])

This type of shifting is repeated in the next phase. In addition, several questions arise about *conceptual* entities and relationships in Figure 2; *e.g.*, why is Alternate airport not a class? Is ATC "part" of Airport? There seems to be a conceptual difference between verb-based relationships (shown by bold lines; *e.g.*, "operates," "is operated by") and other types of relationships (lightweight lines) such as "alternate airports."

The FM diagrammatic representation gives a consistent depiction of the conceptual landscape in the form of spheres with streams of flow that trigger each other, with the switch to technical procedural description, positioned in six stages (boxes in the FM diagram), accomplished in a process like activating hyperlinks in a text.

2.3. Description of choreographies

Chatel *et al.* [13] provide a seminal high-level definition of the rerouting choreographies, based on the BPMN standard:

> The *global* choreography defines all the elementary activities related to dealing with an unexpected flight redirection, including the negotiation of the rerouting between ATC and pilot, passenger information during and after the flight, as well as unexpected arrival handling by airline ground staff [13] (italics added).

Figure 3 shows a partial view of the BPMN diagram of the subchoreography of "Management of unexpected arrival." Such a *global* view is needed after UML-based scenarios and diagrams are developed. Also, note the abrupt transition caused by using a different style of modeling and diagramming at this stage of the project.

In FM, the global conceptual view is developed first and used as a base for further details and stages of design, as in first raising a tent, then returning to set its interior and exterior details.

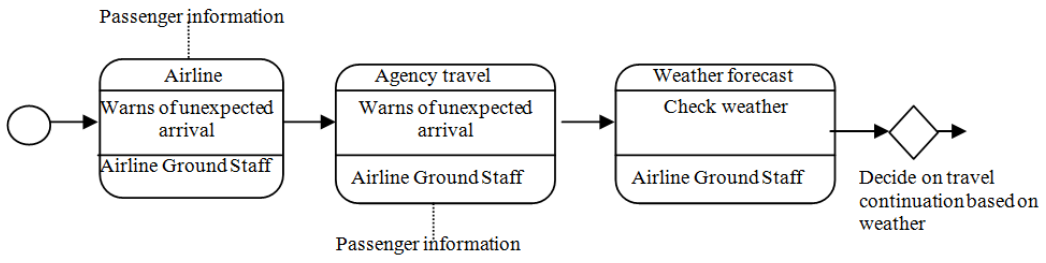


Figure 3. Partial BPMN diagram of scenario 2, "Management of unexpected arrival" subchoreography (from [13])

In preparation for contrasting this approach with FM, and to make this paper self-contained, the next section briefly reviews the Flowthing Model, as described in several publications (*e.g.*, [14]-[17]).

3. Flowthing model

The Flowthing Model (FM) depicts the way a system is structured by providing a roadmap of its components and conceptual flow. Components are termed *spheres* (*e.g.*, a company, robot, human, assembly line, station) that may enclose or intersect with other spheres (*e.g.*, the sphere of a house contains rooms, which in turn include walls and ceilings). Or, a sphere embeds flows (called *flowsystems*; *e.g.*, walls encompass flowsystem subspheres such as pipes of water flow and wires of electrical flow). Things that flow in a flowsystem are referred to as *flowthings* (*e.g.*, money, data, products, cars, parts). The lifecycle of a flowthing is defined in terms of six mutually exclusive *stages*: creation, process, arrival, acceptance, release, and transfer. Within a certain sphere:

- *Creation* means the appearance of the flowthing in the totality of the system of a sphere for the first time (*e.g.*, creation of a new user in a computer system).

- *Process* means applying a change to the form of an existing flowthing.).
- *Release* means marking a flowthing "to be output" while the flowthing remains in the sphere (*e.g.*, a product is marked "to be shipped" but has not yet left the flowsystem).
- *Transfer* denotes the entry of the flowthing into the input/output module of the flowsystem (*e.g.*, an interface component (port) of a device which is part of the device yet is the "door" to the communication channel).
- *Arrival* means that the flowthing reaches the sphere but is not necessarily allowed to progress inside it (*e.g.*, a letter addressed to the wrong recipient that is in his/her hand but will be returned unopened).
- *Acceptance* means permitting the arrived flowthing to enter the flowsystem.

Figure 4 shows a flowsystem with its stages, where it is assumed that no released flowthing flows back to previous stages. The reflexive arrow indicates flow to the Transfer stage of another flowsystem. For simplicity's sake and where appropriate, the stages Arrive and Accept can be combined into one stage termed *Receive*.

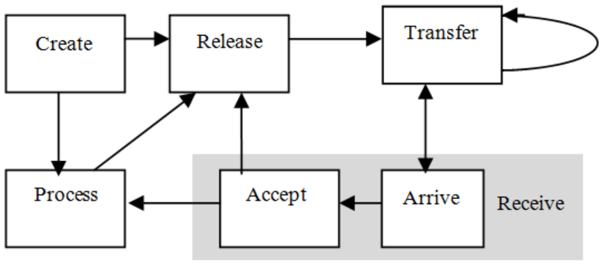


Figure 4. Flowsystem

Some of the verbs used in the UML diagram of Figure 1 can be categorized according to Figure 4:

- "inform" means to create, release, and transfer a message
- "transport" means to release and transfer a flowthing
- "warn" means to create, release, and transfer a warning
- "manage" identifies a sphere of activity that includes a subsphere for release and transfer of passengers

The *stages* in the lifecycle of a flowthing are mutually exclusive (*i.e.*, the flowthing can be in one and only one stage at a time). All other stages, or states, of flowthings are not generic states. For example, we can have *stored* created flowthings, *stored* processed flowthings, *stored* received flowthings, etc. The stage denotes all equivalent generic actions, *e.g.*, create means generate, appear (for the first time), input; *i.e.*, a flowthing appears in a system for the first time.

In addition to flows, *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. Triggering in the case of a system with a single flowthing is a mechanism to change the flow such that releasing/transferring a flowthing *triggers* the creation of another flowthing, *e.g.*, a program that continuously generates (creates), releases, and transfers random numbers one at a time:

```
CREATE: GENERATE A RANDOM NUMBER
RELEASE
TRANSFER:
          OUTPUT
          TRIGGER CREATE
```

assuming that the random number flows through the three stages with no constraints. By contrast,

```
CREATE: GENERATE A RANDOM NUMBER
RELEASE
TRANSFER: OUTPUT
```

does not disturb the flow with a triggering mechanism, and generates continuously regardless of the numbers currently in the flowsystem. The numbers may be queued in the creation or released stages.

Triggering is also a mechanism in which a flow of one type of flowthing triggers the flow of another type, *e.g.*, electricity triggers a light to switch on. Triggering can also be used to activate a sphere. After all, a *state* of a system is a flowthing that can be created, *e.g.*, ON (create light), or OFF (de-create light).

4. Applying FM to the "Bad weather at destination" scenario

The approach summarized in section 2 reflects multilevel modeling that starts with the UML-like activity diagrams of scenario "Bad weather at destination", then adds step-level requirements (class diagrams) with more detailed specifications of the scenario, and finally a choreography description that involves BPMN diagrams. The FM approach aims, from the beginning, at drawing a conceptual flow map, shown in Figure 5, that forms a representational base for any further specifications such as constraints, synchronization, logical operations, and even building a user interface. The figure is drawn based on an understanding of the given description of "Bad weather at destination"; nevertheless, if there is some deviation from this understanding, the diagram can be easily modified. In this explanation of the details of the diagram, numbers in circles are used to guide the reader to different points in the diagram.

The diagram includes three "types" of spheres: the closed airport (top), the alternative airports (middle), and the airlines (bottom right). The process starts when news of bad weather arrives at airport administration (circle 1) to be processed (2), triggering (3) the creation (4) of a decision to close the airport. We assume that this decision is made in the administration subsphere, which includes two flowsystems: Weather information and Decision. In the UML activity diagram (Figure 1) this process is indicated by the box "Close airport runways" inside the dashed box labeled Rerouting. There the focus is on activities without regard for who is doing these activities, assuming that another UML diagram (*e.g.*, class diagram) will include this dimension of the conceptual picture. Yet, this line of modeling is not followed systematically since other activities such as "inform" are included in the boxes of the entities that perform them, such as *ATC informs* and *Pilot informs*.

In Figure 5, the created decision flows (5) to ATC, where it is processed (6) to trigger the creation (7) of a modified flight plan for each flight. The flight modification is a flowthing that flows (8) to the relevant pilot, who transfers it (9) to the passengers (10) and the airline (11). It is not clear how the pilot sends this information to his/her airline, but we assume that such information is entered into the airline database (bottom right box in the figure).

The entire process as conceptualized by FM comprises a continuous flow of weather data to the proper party in airport administration who creates, when appropriate, a decision to close the airport. The "control" in this scenario is formed by the flow of data that triggers a decision that in turn flows to ATC, … just as falling domino pieces transfer their movement to each other in a cascade.

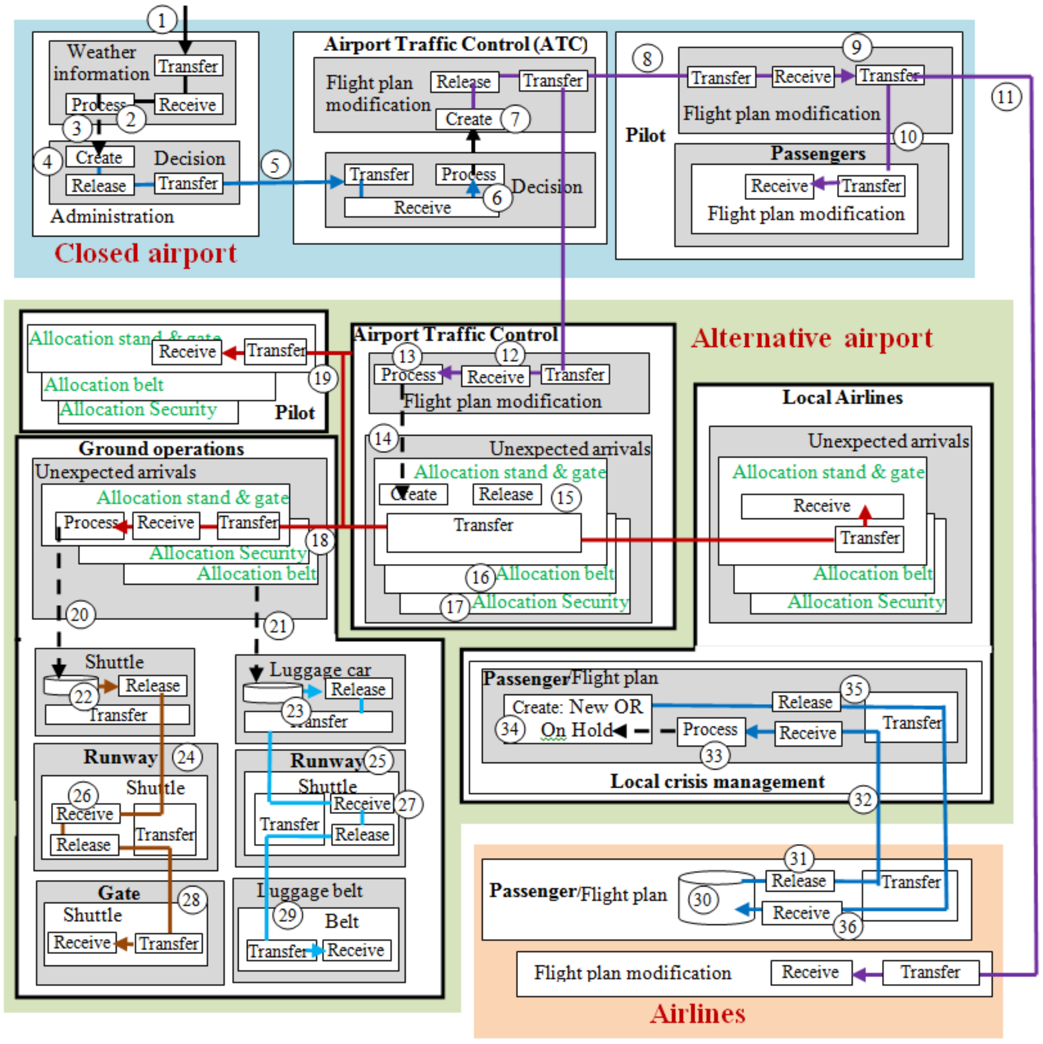


Figure 5. FM representation of the "Bad weather at destination" scenario

This FM representation contrasts with the activities-based process in the UML activity diagram (Figure 1) where "close airport runways", "request", "reroute", "inform",… imply a central "control" that issues these commands. A UML activity diagram is a type of "flowchart" originally developed for programming languages, whereas FM is a conceptual map of movement of different flowthings in their chronological order; however, in FM, the flowchart-like instructions can be embedded into different flowsystem stages. For example, the *process* box of weather information (circle 2) can embed automatic declaration of a decision to close the airport, *e.g.*,

IF (WEATHER DATA IS "SEVERE STORM" AND APPROVAL OF ADMINSTRATOR) THEN *TRIGGER CLOSURE-OF-AIRPORT*

In this case *TRIGGER CLOSURE-OF-AIRPORT* is the flowsystem *Decision* in Figure 5 that comprises creation of the decision to close runways and release/transfer of this decision to ATC. Thus, the FM modeling of "activities" inserts them into their respective contexts (subspheres) categorized according to stages. Accordingly, UML activity diagrams are one way of detailing the interiors of stages of flowsystems in FM. The FM description is a higher level topography of a conceptual map.

Assume a *single* flowthing; just as in drawing a topographic map of a river basin, an FM portrait is formed from:

- Hierarchies of spheres/subspheres, analogous to countries/subregions and valleys in which the river runs.
- Different stages, up to six (creation, processing, …) in which the flowthing can be, according to the flow map, *e.g.*, *creating* (random) binary numbers that are *processe*d to be converted to decimal numbers and *released/transferred* to a printer. This is analogous to the stages of water in subregions; *e.g.*, in the north region, snow is *created* (formed) that is *processed* (transformed) to become slush that moves (release/transfer) to the south region where it is *processed* by climate to become muddy water…
- Different operations on the flowthing when it is in a certain stage (not shown in Figure 5) (*e.g.*, creating a random binary number using different algorithms), analogous to the operations on water in a certain state (stage), *e.g.*, processing muddy water to make it clear.

Going back to Figure 5, at each alternative airport, its ATC receives (12) the *Flight plan modification* from the closed airport and processes it (13) to trigger the *state* of *Unexpected arrivals*, which includes Allocation stand & gate (15), Allocation belt (16), and Allocation security (17). Note that the ATC of the alternative airport has two subspheres: *Flight plan modification* and *Unexpected arrivals*. The subspheres can be interpreted as two different "work situations" that involve the ATC:

(1) Handling (receiving/processing) the data of *Flight plan modification*, in which ATC plays the role of a receiving partner in a communication.

(2) Handling *Unexpected arrivals*, in which it plays the role of supervisor of this situation, generating allocation data and sending the data to *Ground operations* (18), *Pilot* (19), and *Local Airlines*.

At this point, the activity diagram of Figure 1 lumps things together in such statements as "Manage stand and gate process (transport passengers to gate, shuttle, stairways, boarding bridge, *etc*.)." This makes the UML diagram inconsistent in representing activities in comparison with representations of previous activities. It creates gaps in the chronology of events, just as it would if a movie went into detail in a scene, then abruptly ended by listing what happens next: event2, event3, event4,... The topographic nature of the FM representation helps by continuing to draw *Allocation stand & gate*, *Allocation belt*, and *Allocation security*, specifying some portions as follows (other details can be modeled in a similar way):

The instruction to *Allocate stand & gate* triggers (20 and 21) the movement of shuttles (22) and luggage cars (23) to the runway (24 and 25). Note that the shuttle and luggage car are in two contexts (subspheres): *Ground operations* and *Runway*. At this point it is implicit that when the shuttle and luggage cars are on the runway (26 and 27), they are carrying the passengers and luggage. Figure 5 also does not include such final details in the diagram, like a movie suddenly going from a scene of empty shuttles and luggage cars driving onto the runway to a scene of them returning filled with passengers and luggage. The notion of *flow* facilitates the background that fills in the sequence of the scene. Accordingly, the shuttle flows to the gate (28), and the luggage car flows to the belt entrance (29).

"Local crisis management" in the UML activity diagram (Figure 1) is also textual and lumps details together: *The Airline ground staff makes a decision between putting the passengers on hold (for the night) until the weather conditions are restored, or directly transporting the passengers to the originally planned destination (using new flight, bus, and/or rail)*. In FM, in interpreting this description, it is assumed that "Local crisis management" involves accessing the airline database (30) for the flight plan of each passenger and updating it with new data. The record of each passenger is retrieved from the database (31) to flow to Local crisis management (32), where it is processed (33) to trigger the creation of a new record of flight plan or putting the passenger On Hold (34). The resultant record then flows to update the airline database (35 and 36).

5. Conclusion

With the turn of the 21st century, the UML-based conceptual modeling process [18] seems to be the present and future approach enriched by new methods that emphasize integrated methods and new applications with modifications that tune its general features, e.g., orchestrating vs. choreography. For example, in service-oriented systems, it is claimed that the "CHOReOS platform has the potential to become the basis for the development of new consumer habits and radically change the way we conduct everyday transactions as we know today" [19].

Nevertheless, some attention ought to be given to rethinking alternative paradigms, even at the basic representation level in the field. FM is a scheme in this direction that can enhance the dominant modeling technique. In this paper, the discussion has taken the form of redoing an already modeled example to facilitate a side-by-side comparison of the two techniques. The discussion in the paper highlights some of the features of the two approaches; however, comparison of the overall FM representation of Figure 5 with the diagrams in Figures 1, 2, and 3 conveys a general judgment of the features of both methodologies.

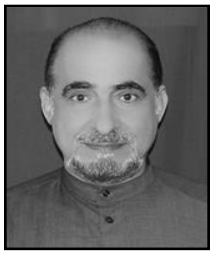
In conclusion, the claim in this paper is that FM presents a viable approach to modeling that can advance knowledge and skill in developing conceptual models that serve as foundations for depicting the relevant processes and relationships in systems.

Whereas this paper focuses on the UML-based approach, FM can be applied to most current diagramming methodologies (*e.g.*, Architecture description languages (ADLs) [20]) used as tools in high-level specifications for information systems.

References

[1] M. P. Papazoglou and W. J. Heuvel, “Service Oriented Architectures: Approaches, Technologies and Research Issues”, VLDB J., vol. 16, **(2007)** July, pp. 389–415.
[2] E. Aviles-Lopez and J. Garcia-Macias, “TinySOA: A Service-oriented Architecture for Wireless Sensor Networks”, Service Oriented Comput. Appl., vol. 3, no. 2, **(2009)**, pp. 99–108.
[3] C. Peltz, “Web Services Orchestration and Choreography”, Computer, vol. 36, **(2003)**, pp. 46–52.
[4] CHOReOS Project Team, “CHOReOS State of the Art, Baseline, and Beyond”, Public Project deliverable D1.1, **(2010)** December 31.
[5] M. Autili, *et al.*, “CHOReOS Perspective on the Future Internet and Initial Conceptual Model”, Project report, version 1, (2012) January 27, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.220.3072&rep=rep1&type=pdf.
[6] N. Georgantas, G. Bouloukakis, S. Beauche and V. Issarny, “Service-oriented Distributed Applications in the Future Internet: The Case for Interaction Paradigm Interoperability”, ESOCC 2013: European Conference on Service-Oriented and Cloud Computing, **(2013)**.
[7] CHOReOS, Plog, **(2013)**, http://www.choreos.eu/bin/view/Main/.
[8] Business Process Model And Notation (BPMN) Version 2.0, Object Management Group, **(2011)** January.
[9] OW2 Consortium, “CHOReOS, Large Scale Choreographies for the Future Internet, from theory to practice”, OW2con’12, Paris, Presentation, **(2012)**, http://www.slideshare.net/OW2/choreos-ow2con12-paris.
[10] D. Silingas, “Model-Driven Service Choreographies for the Future Internet”, Future Internet Symposium 2012, Vilnius, Lithuania, **(2012)** May 21-23.
[11] M. A. Gerosa and C. E. M. dos Santos, “Enabling Scalable Cloud Service Choreographies”, Presentation, http://ccsl.ime.usp.br/baile/files/HPLabs_Gerosa&Cadu_0.pdf.
[12] J. Vanhatalo, H. Vlzer, F. Leymann and S. Moser, “Automatic workflow graph refactoring and completion”, Service-Oriented Computing, ICSOC 2008, ser. LNCS, Springer, vol. 5364, **(2008)**, pp. 100–115.
[13] P. Chatel, A. Leger and J. Lockerbie, “Large Scale Choreographies for the Future Internet: Requirements and scenarios for the "Passenger-friendly airport"”, **(2011)** October 1, http://www.choreos.eu.
[14] S. Al-Fedaghi, “Flow-based Enterprise Process”, Int. J. Dbase Theory Appl., vol. 6, no. 3, **(2013)**, pp. 59-70.
[15] S. Al-Fedaghi, “Toward Flow-Based Semantics of Activities”, Int. J. Software Eng. Appl., vol. 7, no. 2, **(2013)**, pp. 171-182.
[16] S. Al-Fedaghi and B. Al-Babtain, “Modeling the Forensics Process”, Int. J. Security Appl., vol. 6, no. 4, **(2012)** October.
[17] S. Al-Fedaghi, “Developing Web Applications”, Int. J. Software Eng. Appl., vol. 5, no. 2, **(2011)**, pp. 57-68.
[18] J. Evermann and Y. Wand, “Towards ontologically based semantics for UML constructs”, in H. Kunii, S. Jajodia, and A. Solvberg, Editors, Proceedings of the 20th International Conference on Conceptual Modeling, **(2001)**, Yokohama, Japan. http://www.mcs.vuw.ac.nz/ ~jevermann/EvermannWandER01.pdf.
[19] CHOReOS, Plog, The February 2013 interview with George Veranis. http://www.choreos.eu/bin/view/Follow/Interviews-February-2013.
[20] W. Jing, Y. Shi, Z. LinLin and N. YouCong, “AC2-ADL: Architectural Description of Aspect-Oriented Systems”, Int. J. Software Eng. Appl., vol. 3, no. 1, **(2009)**.

Author



Sabah Al-Fedaghi 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 170 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).

http://cpe.kuniv.edu/images/CVs/sabah.pdf