Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 20 (2013) 270 – 276

Procedia Computer Science

Complex Adaptive Systems, Publication 3
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2013- Baltimore, MD

# Behavioral Modeling of Software Intensive System Architectures

Monica Farah-Stapleton[a*], Mikhail Auguston[b]

[a] *Interagency Program Office, OSD, 1700 N. Moore St , Rosslyn VA, USA*
[b]*Department of Computer Science, Naval Postgraduate School, Monterey, CA, USA*

**Abstract**

Architectural modeling and analysis are mechanisms that allow the capture of design decisions early in the process, so that they can be assessed and modified without incurring the costs of incorrect implementations. This paper addresses Monterey Phoenix (MP), a behavioral model for system and software architecture specification based on event traces, which supports architecture composition operations and views. MP captures behaviors and interactions between parts of the system and the environment with which it operates. As an executable architecture model leveraging "lightweight" formal methods and the small scope hypothesis, MP supports automatic generation of behavior examples (Use Cases) for early system architecture analysis, testing, verification, and validation. This paper also introduces a methodology utilizing MP that will inform quantifiable cost estimates (e.g. Function Point analysis) and ultimately project, program, and enterprise level resourcing decisions. Enhancing and extending DoDAF, UML, and SysML, MP is focused on behaviors, interactions, and automated tools for early verification.

Keywords: architecture, behaviors, function point analysis, event traces

## 1. Introduction

Within the Department of Defense (DoD), there have been significant but often disconnected efforts to develop architectural descriptions of systems and the environments in which they operate. Complex architectural design decisions are often captured on a system by system basis, using a spectrum of representations from natural language to formal notations. These inconsistent representations are then analyzed through manually intensive methods such as inspections and reviews. The result is system and software architecture and development efforts that are unrelated or duplicative, with a technically and programmatically unsustainable result.

The implication is that modeled architectures are a waste of time and money, which is unfortunate because architectures matter. If developed and utilized properly, these models enable capturing behavior of not only the system, but also of the environment with which it interacts. They can reflect design decisions, provide a framework

* Corresponding author. *E-mail address*: monica.farah-stapleton@tma.osd.mil.

1877-0509 © 2013 The Authors. Published by Elsevier B.V.
Selection and peer-review under responsibility of Missouri University of Science and Technology
doi:10.1016/j.procs.2013.09.272

to reason about those decisions, and then facilitate verifying assertions early enough in the design process to prevent incurring the costs of incorrect implementations. They also matter because accurate and complete architectural descriptions establish a common mental model among stakeholders, helping them to answer groups of questions on topics such as:

- System development strategies and technology insertion.
- Disposition strategies for legacy systems.
- Meaningful engineering metrics that inform forecasting (e.g., estimates of new services or new system elements) of a system's development, testing, and evolution.
- Interoperability and integration strategies that inform Total Cost of Ownership (TCO) and Return on Investment (ROI).

Although not all stakeholders understand the details of modeling, they do understand cost savings, cost avoidance, and efficiencies. They also understand the need for data that will inform their decisions to invest in specific implementations, and allow them to quickly and accurately assess whether the investment is warranted.

Monterey Phoenix (MP) [1][2][3], is a behavioral modeling framework for system and software architecture specification, that supports architecture composition operations and views. An example, derived from the International Function Point User Group IFPUG [4], illustrates that once the behaviors and interactions associated with a system and its environment are understood, familiar estimation practices such as Function Point counting can be used. The proposed methodology employing MP extracts analysis enablers from the model, including Views, Use Cases, and programmatic metrics of schedule, effort, and size estimates.

## 2. Views, Use Cases, and Function Points

Although there are many definitions of the term "system", it can be defined as a whole that cannot be divided into independent parts without losing its essential characteristics as a whole [5]. Systems are comprised of hardware and software, data, procedures, and people interacting with the system [6]. Such complexity requires methods and automated tools to assist stakeholders in extracting meaningful information from the architectures describing the system and environment.

Architectural views and the viewpoints describing them help to portray aspects or elements of the architecture that are relevant to the concerns that the view intends to address, and to the stakeholder interested in those concerns. [7]. Each view is an answer to a question (or a group of questions), and provides the rationale for the development of tools, patterns, templates, and conventions needed to create the level of abstraction that reduces complexity while retaining meaningful content.

Use Case and Function Point (FP) descriptions can be considered ways to view a system, its sub-components and the environment, in order to address the concerns of specific stakeholders. Current techniques utilizing Unified Modeling Language, Systems Modeling Language (SysML), DoD Architecture Framework (DoDAF) [8] views, and FP analysis can be enhanced and related to remove ambiguity and link the descriptions. One approach to relating these techniques is to use the following proposed methodology:

- Describe the behavior of the system and environment in natural language.
- Refine the natural language until the boundary of the system and the environment can be clearly identified.
- Represent the system and the environment in a high-level visualization, e.g. box-and-arrow traditional architecture diagram view.
- Identify the Function Points based descriptions of system components and sub-components, user-system, and system-environment interactions.
- Describe the behavior of the system in MP, refining the model as required to understand the behaviors of the system and environment at the detail necessary to answer stakeholder concerns.

- Extract Use Cases from the MP model and visualize them through event traces and traditional architecture diagrams.
- Continue to refine the MP model and confirm the interactions are represented consistently in the Function Point analysis methodology.

There are mature processes associated with each step of this proposed methodology which are effective for each individual step. By combing these steps, identifying the appropriate level of abstraction, and then extracting the answers to questions or groups of questions from the result, this proposed methodology intends to inform resourcing decisions based on analytical underpinnings and then render familiar views to decision makers.

Use Cases are a means to capture what a system is supposed to do as described by actors and the system under consideration [9]. Sequence diagrams in UML and SysML represent a view of a Use Case, providing a common visualization that assists software engineers, developers, and stakeholders to communicate with each other, and develop a rudimentary understanding of the behavior of the system and the environment.

Function Points are a normalized metric used to evaluate software deliverables and to measure its size based on well-defined functional characteristics of the software system. They must be defined around components that can be identified in a well-written specification [10].

As illustrated in Figure 1, Function Point terminology can be used to describe the interactions of a user, system and its environment:
- External Inputs (EI): Input data that is entering a system.
- External Outputs (EO) and External Inquires (EQ): Data that is leaving the system.
- Internal Logical Files (ILF): Data that is processed and stored within the system.
- External Interface Files (EIF): Data that is maintained outside the system but is necessary to satisfy a particular process requirement.

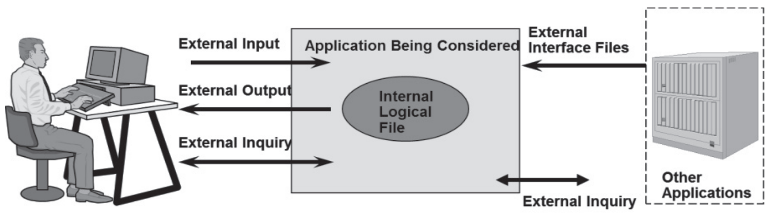

Fig. 1. Functionality As Viewed From The User's Perspective [4].

MP, as a behavioral model for system and software architecture specification based on event traces, supports different architecture composition operations and views. This software and system modeling framework leverages lightweight formal methods to unambiguously describe the behaviors and interactions of a system and its operational environment, capturing design decisions about precedence, inclusion, concurrency, and ordering (dependency relation between activities). As an executable architecture model, it can then be used to automatically generate examples of the behaviors (e.g. Use Cases) from these specifications of behaviors and interactions for early system architecture analysis and testing. MP does not replace system and software engineering enablers such as UML, SysML, and DoDAF 2.0, but complements them, and emphasizes the necessity of automated tools for early verification.

Use Cases, Function Point Analysis, and behavioral modeling frameworks such as MP can help stakeholders understand the technical and programmatic characteristics of the system and environment, by effectively creating views that contain the information they need. The methodology employing MP extracts analysis enablers from the model such as Views, and Use Cases, and informs programmatic metrics of schedule, effort, and size estimates.

## 3. Monterey Phoenix

Simplistically, the behavior of a system can be described in terms of an algorithm, i.e. a step-by-step set of activities for solving a problem or accomplishing some end [11]. MP represents an event as an abstraction of an activity. The behavior of a system can then be modeled as a set of events with two binary relations defined for them: precedence (PRECEDES) and inclusion (IN) – the event trace. Since the event trace is a set of events, additional constraints can be specified using set-theoretical operations and predicate logic. A more detailed discussion of MP grammar rules and examples can be found in [1] and [3].

Consider the following MP schema MP_Function_Points which describes the behaviors of user, system, and environment interaction, as illustrated in Figure 1. The behavior of the user, system, and environment is illustrated as a traditional architecture view in Figure 2. A Use Case (event trace) extracted from the MP model is illustrated in Figure 3, and represents the following behaviors:

Lines 01-03: The User inputs data or submits a query and receives processed data or an error message.
Lines 04-09: The Application_Being_Considered (hereafter referred to as Application) receives account data, checks for data consistency, stores account data, sends processed data.
Lines 10-11: Other_Applications check for consistency of data.
Lines 12-23: Composition operations specify the interactions among the systems.

*Schema MP_Function_Points*

```
01 ROOT User:
02 (*     (input_account_data  | submit_query  )
03                    (receive_processed_data | error_message)     *);

04 ROOT Application:
05 (*  (receive_account_data  | receive_query )  *) ;
06         receive_account_data : check_data_consistency
07         ( store_account_data    send_processed_data |  error_message);
08         receive_query: ( [ request_EIF  receive_EIF ]  process_query )
09         ( send_processed_data |  error_message );

10 ROOT  Other_Applications:
11                    (* ( send_EIF |  check_data_consistency  ) *) ;

12 Application, User SHARE ALL  error_message;
13 Application, Other_Applications              SHARE ALL       check_data_consistency;
14 COORDINATE (* $x:input _account _data *)  FROM User,
15             (*$y:receive_account_data*) FROM Application  ADD $x PRECEDES $y ;
16 COORDINATE (*$x:send_processed_data *)  FROM Application,
17             (*$y:receive_processed_data*) FROM User  ADD $x PRECEDES $y ;
18 COORDINATE (*$x:submit_query*) FROM User,
19             (*$y:receive_query*) FROM Application              ADD $x PRECEDES $y;
20 COORDINATE (*$x: request_EIF *)  FROM Application,
21             (*$y: send_EIF *) FROM Other_Applications  ADD $x PRECEDES $y ;
22 COORDINATE (*$x: send_EIF *)           FROM Other_Applications,
23          (*$y: receive_EIF *)    FROM Application         ADD $x PRECEDES $y ;
```

Function Point analysis is a measurement practice for sizing software. One of the earliest steps in the FP counting process is identifying the counting scope and application boundaries. The methodology employing MP assists in unambiguously identifying the boundaries and interactions of the system, user, and environment. Function Points functions can be thought of as markers of the boundaries. Once the boundaries and interactions have been described, the Function Point analysis practice can be used to size the software including: Counting the Data and Transactional Function Types; Determining the Unadjusted FP count and the Value Adjustment Factor; and then calculating the final Adjusted FP Count.

For this example, the MP model highlights that there are five Function Points functions identified, three between the User and Application, and two between the Application and Other Applications. The model also highlights dependencies between Function Points functions, and that the Error Message Event invokes effort and therefore cost. Table 1 provides an example of the relationship of Function Point terminology to MP terminology, each describing the interactions between the user, system, and the environment.

Table 1. FP to MP Terms Example.

| Component/MP ROOT | FP Description | High Level MP Description |
|---|---|---|
| User | External Input | input_account_data |
| | External Output | receive_processed_data \| error_message |
| | External Inquiry | submit_query |
| Application Being Considered | External Input | receive_query |
| | External Output | receive_account_data |
| Other Applications | Ext Interface Files | send_EIF |
| | External Inquiry | check_data_consistency |

A traditional architecture view of the user, system, environment behavior described in the *MP_Function_Points* schema is illustrated in Figure 2, emphasizing the interaction between the parts components. Solid green lines represent the behavior composition operation COORDINATE. The pink lines represent the behavior composition operation SHARE ALL. Such views can be extracted from the MP model using automated tools.

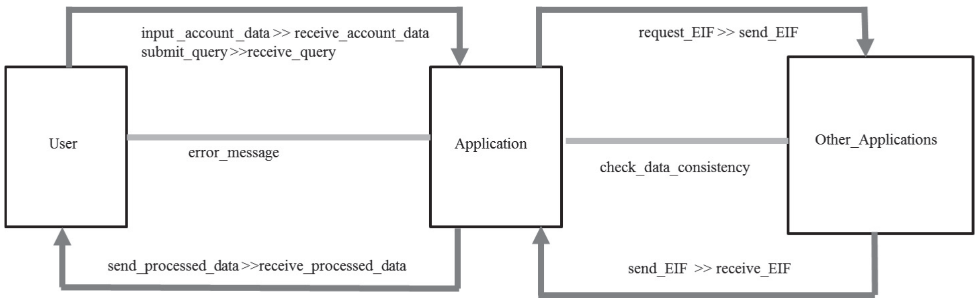
input _account _data >> receive_account_data
submit_query >>receive_query
request_EIF >> send_EIF
User
error_message
Application
check_data_consistency
Other_Applications
send_processed_data >>receive_processed_data
send_EIF >> receive_EIF

Fig. 2. Architecture View of MP_Function_Points schema.

Figure 3 illustrates a view of an event trace, i.e. one Use Case, that has been extracted from the MP_Function_Points schema. There are two levels of IN from the Application root event: receive_account_data as a composite event under the IN relation with the Application root event, and with the other events IN receive_account_data. Other_ Applications is another root event.

If the view of the system's behavior emphasizing the interaction between the parts (components) and the environment can be visualized as in Figures 1, 2, and 3, MP models can be integrated into standard frameworks, like UML, SysML, and DoDAF, providing the level of abstraction convenient for architecture models. Employing MP to capture behaviors and interactions, and then expressing them in a format that is more familiar to the user, transforms a purely academic investigation into a practical exercise to capture high level design decisions, enable pattern identification and reuse, and quantify cost avoidance, savings and ROI.

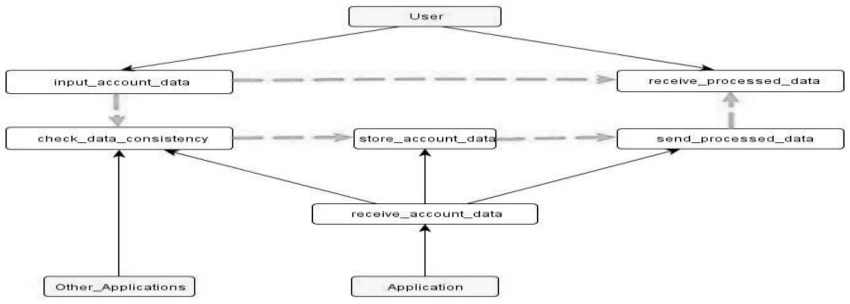

Fig. 3. Event Trace (Use Case) extracted from MP_Function_Points schema.

## 4. Summary and follow-on work

Architectural modeling and analysis allow reasoning about the behavior of systems and environments. Monterey Phoenix, as a behavioral model for system and software architecture specification based on event traces (Use Cases), leverages lightweight formal methods to unambiguously describe those behaviors. MP has powerful interaction abstraction. Separation of the behavior of the component from the interaction between components is an important feature for model reuse. Additionally, the MP assertion language is very expressive, so it is more feasible to perform various computations on specific instances of event traces. MP is focused on "lightweight" verification, i.e. exhaustive trace generation, based on the Small Scope Hypothesis (executable architecture models) [3]. Use Case and Function Point descriptions can be considered ways to view a system and the boundaries of its sub-components and the environment, in order to address the concerns of specific stakeholders. The proposed methodology employing MP can be applied to classes of tasks associated with hierarchical interactions between a system and its environment. Refining and evolving this methodology will include: Identifying behavioral patterns for system-environment interactions; Determining what behaviors to abstract and what questions or groups of questions can be addressed; and Considering how visual representations, automated tools, and automated estimation methodologies can inform technical and programmatic decisions.

## References

1. M. Auguston, 2009, Software Architecture Built from Behavior Models, ACM SIGSOFT Software Engineering Notes, 34:5.
2. M. Auguston, C. Whitcomb, 2010, System Architecture Specification Based on Behavior Models, in Proceedings of the 15th ICCRTS Conference (International Command and Control Research and Technology Symposium), Santa Monica, CA, June 22-24, 2010.
3. M. Auguston, C. Whitcomb, 2012, Proceedings of the 24th ICSSEA Conference (International Conference on Software and Systems Engineering and their Applications), Paris, France, October 23-25 2012.

4. International Function Point User Group, inspired by "Introduction to the International Function Point Users Group (IFPUG), p.6, © Copyright 1999, International Function Point User Group 1999.
5. INCOSE Systems Engineering Handbook version 3, 2006, p2.2.
6. N. Rozanski, Software Systems Architecture, Addison Wesley, 2012.
7. R. Taylor, N. Medvidovic, E. Dashofy, Software Architectures, Foundations, Theory, and Practice, John Wiley & Sons, 2010
8. DEPARTMENT OF DEFENSE, 2009, DoD Architecture Framework, Version 2.0, Washington, DC: ASD(NII)/DoD CIO, 2009.
9. Object Management Group Unified Modeling Language (OMG UML), Superstructure, May 2012, ISO/IEC 19505-2:2012 (E), April 2012.
10. Object Management Group, Automated Function Points Specification, FTF Beta 1, Feb 2013.
11. Merriam-Webster, http://www.merriam-webster.com/dictionary/algorithm.