The Research Monograph Series in Computing, Electrical & Communication Networks
Vol. 1 (01), April 2023, pp. 1–7

CHAPTER 1
INTRODUCTION

Over the decades many advanced systems have been created by the mankind through the inventions and the applications of new technologies System engineering gains a broader discipline due to the emergence of new inventions and technologies that creates new engineering disciplines. Understanding the fulfillment of performance requirements is the fundamental thing in the software development process, which represents the expectation of end users from the software system. Otherwise this results in critical consequences of the system. The phases with wrong decision at early development phases, heavily affect the quality of the final software product, they may demand for an expensive rework and it may include the involvement of overall software systems. In order to avoid the failure of the entire projects, the performance issues must be identified early at the right time.

The earliest model is the software architecture of a software system, according to Perry and Wolf (1992), architecture refers to the selected architectural elements, their interactions and the constraints on those elements. This thesis follows the views of computational components (Koziolek A. and Trubiani C., 2011) with thin description of interactions.

Predicting quantitative results has been the basis for the applications of several successful approaches through performance of software systems.

1.1 SOFTWARE PERFORMANCE ENGINEERING

Over a decade new types of approaches have been applied to take the problem of modeling and qualifying the software performance right from the beginning of the life cycle, as automation has gained the prediction role (Hauck M. et al., 2009) in generation of performance models from software artifacts.

By changing both the structure and the behavior of a system, the new architecture is obtained. More particularly the model solution suggests specific replacement of existing software units with different available instances to modify the system structure, when it becomes necessary of new software units. The model solution suggests how to bring out and apply the changes in the system behavior through the system scenario's expressed using UML (Grady B. et al., 1999) sequence diagrams and by carrying out the removal or introducing interactions between existing units and the new units. Depending on the adopted model notation, of the application domain, environmental constraint etc, different elements stand as the base for the strategies during the identification of performance problems.

1.2 ARCHITECTURE OF THE SOFTWARE

The crisis of software architecture, (Garlan D. and Perry, 1994), as a foundational concept for the development of large complex systems, support five aspects of the software development in respect to understand, reuse, evolution, analysis and management. The most primitive model of a software system created along the lifecycle is software architecture. According to Perry and Wolf (1992) the architecture is selection of architectural elements which include their interactions all with the constraints on those elements. In the view of Garlan and Shaw (2003) the architecture is defined as a combination of the collection of computational components (Koziolek H. et al., 2008, 2010) and the description of their interactions, of these this thesis adopts the latter. While the abstract view of the software system is provided by a software architectural model (Aldrich J., 2008), different system information is provided by the complementary types of model. A software development model is shown in Figure 1.1. These different models are presented through various perspectives focusing on the behavior of the system, external perspective based on the system's context.

1.3 SOFTWARE DEVELOPMENT PROCESS

The increase in terms of size, logical distribution and the complexity of interactions in software systems out phases the growing importance of early performance assessment. From the early phases of the lifecycle an integrated software process cannot be built by the software developers who find no time and create distance among software model notations and performance model representation (Averill M.L. and David Kelton W., 2000). The distance between the world view adopted by software developers and performance experts is one of the issues that prevent the performance validation from being a common practice in the software lifecycle. Generally static and (Bahsoon R., 2007) models with functional aspects are used by software developers to describe a system. Keeping

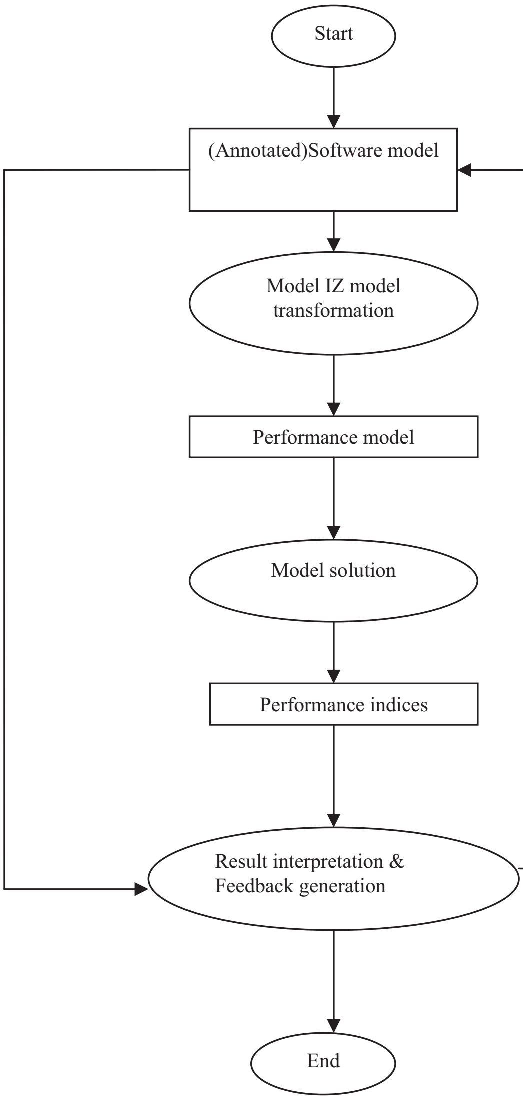
Start
(Annotated)Software model
Model IZ model transformation
Performance model
Model solution
Performance indices
Result interpretation & Feedback generation
End

Figure 1.1 Software Development Models.

in view of deserving meaningful performance models, performance experts show additional interest in non functional aspects, such as the operational profile i.e., the estimation of execution probabilities of different (Mirandola R. and Hollinger D., 1997) software systems need to integrate software models.

UML is temporarily accepted as a standard for designing new systems (Yuanfang C. et al., 2006) helps system designers with its array of notations to capture their ideas and to make the ideas early understandable and expressive. UML (Alawneh L. et al., 2006) has the limitation, although it is a means of predicting the systems performance directly. Prediction of performance as a feature helps to decide the worth of implementation of a particular design.

1.4 UML

Improving the ability to evaluate software and system design for non-functional properties like performance, reliability and security has been emphasized (Arief L.B. and Speirs N.A., 2000). This evolution can be made suitable for additional information, annotation and attached to the design. Performance and schedulability standard (Object Management Group, 2002) has been addressed in the standard UML profile for schedulability performance and time. (Harel D. et al., 2004).

Capturing the proven architectural design patterns (Smith C.U. et al., 2003) of the domain stand as first class modeling constructs and they are highly useful facilitating the design of good architectures. They are two types of fundamental complementary diagram provided by UML for capturing the logical structure of systems. They are class diagrams and collaboration diagrams (France R.B. et al., 2004). Universal relationships among classes, existing among the instances of the classes in all contexts, are captured by class diagrams. A strong emphasis is laid on the usage of collaboration diagrams in this modeling approach wherein the mediators between architectural entities are explicitly represented by similarity, a combination of class and collaborative diagrams facilitates to obtain the complete specification of the structure of a complex real time system (Henrik Ejersbo Jensen et al., 2000). The three principal constructs for modeling structure are specially defined as

- Capsules
- Ports
- Connectors

1.4.1 Capsules

One or more signal-based boundary objects (Aldrich J. et al., 2008) are called ports through which capsule with the outside world and it is physical part of the implementation of a capsule with a specific interface as shown in Figure 1.2. Each part of a capsule plays a particular role in collaboration with capsules other object, which are associated with a protocol to capture

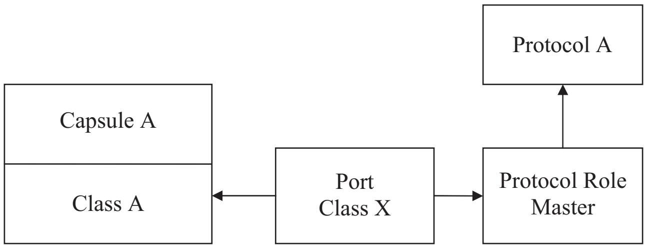

Figure 1.2 Allocation of ports, protocols, and their roles.

the semantics of these interactions. The valid flow of information between connected ports of capsules is defined by the protocol.

1.4.2 Ports

The purpose of ports which are objects act as boundary objects for a capsule instance. Ports, owned by the capsule instance are created and destroyed along the capsule. Thus each port creates its own identity to prove the ports, which are distinct from the identity and state their owning capsule instance.

1.4.3 Connectors

Two or more ports are interconnected by the connectors which are abstract views of signal-based communications channels. It is a must for the ports bound by a connection to play mutually complementary but comfortable roles in a protocol. Their representatives are exhibited by association roles that interconnect the appropriate ports in collaboration diagrams. The key communication relationship between capsules can be really captured by connectors (Arief L.B. et al., 2000) when the ports are removed from this picture.

1.5 FEEDBACK

The interpretation of performance analysis results and the generation of architectural feedback can be forced in the literature related works. These are mostly based on monitoring techniques which are conceived to act for tuning its performance after software deployment. We have applied a model based approaches in the software life cycle support of design decisions.

Various approaches have successfully solved the difficult task of transforming software models into performance model during the last decade, but evaluating and developing the performance of a software system is still being a research issue. Here we have proposed a novel method to develop a performance model based on the design of software which follows forward path and backward path. A forward path is defined for the software model to identify the performance indices, represented by the modeling and analysis phase.

1) The Performance Indices are the numeric values that are linked to model entities which examine the problems in the approach.
2) Various levels of granularity project the performance indices with the model evaluations as outcome to maintain and manage indices abstractions (Sabetta A. et al., 2005) at all levels.
3) The involvement of various characters of software system like static structure dynamic behavior and cross checking of the characters raise the performance problem that makes the software models difficult.

Our work presents a UML profile represents the role of an attribute as an operation, a class that plays in a design pattern and distinguishes multiple instance of a design pattern. As it is already noted, UML is used to capture these constructs proved, the absence of extendible constructs need to introduce new UML modeling concepts.

1.6 PROBLEM DESCRIPTION

An abstract view of the software system is presented through the software architectural model. Different system information from different perspectives is presented by different complementary types of models, e.g. systemic context (Harman M. et al., 2009) of environment through external behavior of the system to show behavioral perspective. Annotated models are referred as they add information to execute performance analysis e.g. incoming work load is the system service demands, software characters etc.

1.7 RESEARCH OBJECTIVE

The thesis will exactly

- Recognize about the translation of software model into performance model.
- Analyze the software model with respect to the performance indices.
- Implement an automatic transforming software model.

In order to attain this entire goal, this research has developed a framework which has the capability of refactoring (Han J. et al., 2007) the software architectural design to give a better performance with respect to the performance indices, by evaluating them at the design phase of software development.

The sub objectives of this research are:

- Analyzing and study about UML
- Research about the software development architecture.

1.8 RESEARCH APPROACH

This research study as in Figure 1.3 starts with the field of automatic transforming software model under the literature study (A) In which UML model software architecture has been used to design the automatic transforming software models found during this literature study (B) and implement the automatic transforming software model with feedback system (C) from the design.

The problem of automatically transforming software artifacts into performance models has been overcome successfully in the last decade by using various approaches (Yuanfang C., 2006).

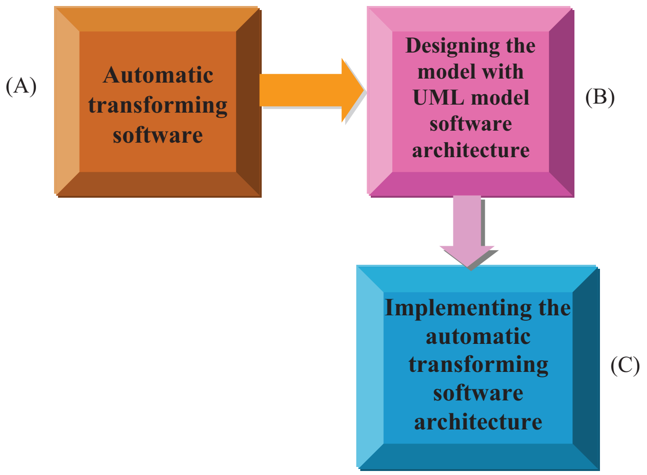
(A)
Automatic transforming software
Designing the model with UML model software architecture
(B)
Implementing the automatic transforming software architecture
(C)

Figure 1.3 Research Approach.