A General Framework to Bridge the Gap Between Conceptual System and Abstract System in Software Development

Mohamed Ali Elshaari
Mohamed Ali Hagal
Zainab Saad Elbadry
A GENERAL FRAMEWORK TO BRIDGE THE GAP BETWEEN CONCEPTUAL SYSTEM AND ABSTRACT SYSTEM IN SOFTWARE DEVELOPMENT


*Faculty of Information Technology, Garyounis Univ. Benghazi, Libya
mohamed.shaari@gmail.com, Mohdhg@yahoo.com

**Faculty of Education, Garyounis Univ. Benghazi, Libya

Abstract:
The goal of software development is to construct systems that can be implemented on computers, but that faces many obstacles, the most important one is that the analyses and design are hard and sometime impossible to implement to computer. This paper is designed as attempt to define an analytical framework, which can synchronize the work between the computer and this problem. It uses general systems theory and formality to build a formal understanding conceptual system for the aimed problem, and then build software architecture interpreted in a formal language which it expresses the components of software architecture. Therefore, the work is designed to deduce that problem. It may be solved if the framework is followed; the computable system reached by using formal language. The computable system supposed to have the ability to be implemented to a computer. A relevant example is a case study about an Electronic Market, which gives an image that clarifies how to follow the framework to solve a software development problem.

Keywords: Abstract Level, Formal, Architecture, OWL, Problem, Ontology, UML.

1. INTRODUCTION

Software development methods are aimed for the development of software systems. Most of them follow a software life cycle, which include several stages; starting by defining the problem, passing to analysis, design stages, and finishing with implementation. However, software development can be faced with many obstacles when trying to implement the designs, which indicates that the design may not be interpreted in a programming language. This inability of interpretation can be caused by work in early stages isolated from late ones, i.e. there is a gap between implementation, analysis and design stages. The reason behind this gap is that the requirements and designs are not formally understandable; which this means the designs could not be interpreted in a formal language to express such situations. Formal languages include logics, mathematics, programming languages, ontology languages and some design languages, etc. Using proper formal languages a formal design, this could reach a high degree of automation when implementing the system.

2. ANALYTICAL FRAMEWORK

To overcome the mentioned obstacles, we built a theoretical framework that could be followed to construct a computable system, which can operate on a machine. The built framework was based on the idea that when developing a software system we should work in two different levels; conceptual level and abstraction level. The first level is concerned with the system that is built to represent the problem. It is based on systems theory used to construct system architecture and models that illustrate how it works. The second level (Abstraction level) is concerned with the machine (i.e. computer) which its software closed with formal languages. The framework concentrates on the first level and keeping an eye on the second to follow the formality conditions, the model emerges and matches a computational notation, i.e. It is programmable.
This level emulates the problem. It is considered descriptive and constructive, but it is not computable. In this level, the components of the software system structure are constructed and the relations among them are set.

COMPONENTS OF CONCEPTUAL LEVEL:
I. DEFINING THE PROBLEM

Describing the problem using natural language to explain its structure, and trying to convert it to components. We should identify and specify any properties in the description, in every component; the relations should be defined between entities.

II. META SYSTEM

One of the difficulties that we are faced with, at the abstract level, is to find a proper formal language that can represent a problem. Hence, a system was built to represent the problem. This system was based on human understanding for the problem and general systems theory. Moreover, the system form which consists of components connected by functional relations governed by rules. It has configurations to imitate all possible situations that could be translated to formal language.

III. CONCEPTUAL SYSTEM

To construct the conceptual system the domain of the problem should first be specified, and the Meta system built. There are no obligatory methods or rules; however the most important necessity is answering this question: "Is the constructed system formally understandable?" Then we could use some formally understandable diagrams and models such as ERD, Use case maps (UCM), state diagram and follow diagram to represent the system.

IV. SOFTWARE SYSTEM ARCHITECTURE

The architecture is considered as a design with more features, the architecture also illustrates the design context and implementation code. It clarifies the system structure, behavior and performance through the dynamic description for the system. This illustrates all system configurations and how they change from one state to another. The description should not involve any components that cannot be interpreted on a programming language. If the Architecture does not provide any computational description, it would not achieve any computations. In this stage, the needed steps are specified for the system to attain its goals, and for the components to verify their functions under the rules specifying their relations.

V. MODEL

All software has a certain architecture, which can be represented by a model, because the model is used to represent a design. First, the system should be well defined. Second, using computational description the model should show exactly how architecture works. Furthermore, it computationally clarifies the dynamic description.

4. ABSTRACTION LEVEL

The computable level emulates the machine and formally represents the conceptual level.

COMPONENTS OF ABSTRACTION LEVEL:
I. FORMALITY

All the components of the conceptual level should be verified, well described and formally understood. This means that the concepts in
conceptual level should involve some specific properties. Formality is a desired property demanded at the abstraction level. If this property is totally verified then we have complete formality, otherwise we have less formality. The properties of formality may be summarized in the following steps:

- The conceptual system components should be transformed to a form that can be dealt with.
- Availability to use symbolic and formal statements to express the content of conceptual level, which permits derivations within the limits of definitions without any obstacles.

II. FORMAL SYSTEM

Formal system is an actual system that involves expressions to convey formality properties. It consists of a set of formal sentences formulated in well-formed formulas by the formal language, and methods that give the capability to use a subset of formal sentences in particular contexts to prove the validity of a group of concepts and relations inside the system. Formal language is a language that is designed to express certain situations which the natural language and ordinary software development methods cannot express. There are several types of formal languages such as Specification languages, Logics, and Mark up languages can be used to represent a system; and there are many formal languages that can be used for specific purposes i.e. Z-notation or VDM (Vienna Development Method) such as writing the specification. These languages transfer the system from systemization and architecture stages to be interpreted in machine i.e. they write the system formally, so the system can be embedded in programming language, but validation of the system is required.

III. COMPUTATIONAL LOGIC

The computational system is a set of tools used to apply formal properties at the conceptual level within the conditions of formal theory. It can construct the formal system by using computational logic, which is generally proper to work on the machine. The working to build conceptual level and abstraction level should be carried out in parallel, since these two levels overlap each other.

The analytical framework which is built in this research is to construct computable systems. It may represent and simulate any well-defined problem. Moreover, the framework illustrates how the components of the conceptual level and abstraction level are overlapped to work together to achieve the task (the goal).

5. CASE STUDY

PROBLEM (E-MARKET):

To order goods from the market, a customer needs to search for goods, place them in a shopping basket, arrange for payment and then forward the order to the market. Each customer order requires packing and pricing.

META MODEL:

In this step, we describe the main functions of the system from the point of view of the system’s external entities. Entities that interact with the system considering the general principles of formality. UML use case diagrams can be used to accomplish this.

CONCEPTUAL SYSTEM:

This step describes scenarios used to demonstrate high-level view of the system, and its flow processes in high-level view. A use case map is used to demonstrate goods purchase scenario.
ARCHITECTURE:

To build architecture of the system, some models of UML was used as examples. A class diagram

as an example that is transformed in the next section into textual description, using formal language, is shown below.

FORMAL (OWL):

This part presents a simple example for representing a sample of the architecture in a selected formal language.

```xml
<owl:NamedIndividual rdf:about="http://example.com/individual1"/>
<owl:NamedIndividual rdf:about="http://example.com/individual2"/>
<owl:NamedIndividual rdf:about="http://example.com/individual3"/>
```

MODELS:

Detailed class diagram is used as an example to represent the clear transformation from the formal description to computational form.

6. CONCLUSION

This paper presented a general framework to simplify the transformation from conceptual level to abstraction level for developing software systems. Formal system design represents the first step to move to machine system. The base of this design is the conceptual design. The goal of this design is to use special tools such as formal language to write the formal description for a compatible system to be implemented in a specific machine. However, not all formal descriptions could be implemented on machine, because a formal language may succeed only in writing some parts of system design and fail in others. The parts successfully written will be implemented in machine. In other words, any system written by using formal logic is not necessarily implemented in machine; more precisely it will not be interpreted in software architecture. Possibly the system is partly unable to be written by a formal language, this inability is related to formal understanding in conceptual design or to nature of the problem, because many problems have no full capability to formalize, hence to computerize.

REFERENCES:


[6] Dean C. Chatfielda, "Corresponding Author Contact Information, E-mail The Corresponding Author, Jack C. Hayyab, 1. E-mail The Corresponding Author and Terry P. Harrison. "Multi-Formalism Architecture for Agent-Based, Order-Centric Supply Chain Simulation." Simulation Modelling Practice and Theory. 02 2007, Vol. 15, 2.


