Knowledge Representation of Acquisition and Control Systems with Graphical Programming using UML Notation

ELVIRA GAYTÁN GALLARDO[1, 2], JAVIER ORTIZ HERNÁNDEZ[3], J. ARMANDO SEGOVIA DE LOS RÍOS[1, 2]
[1]Instituto Nacional de Investigaciones Nucleares
[2]Instituto Tecnológico de Toluca
[3]Centro Nacional de Investigación y Desarrollo Tecnológico
MËXICO

Abstract:- The need to improve the software engineering in the development of complex data acquisition and control systems with reusable components designed with graphical programming languages leads to represent the knowledge acquired in the development of these systems. Software for data acquisition and control systems with graphical programming is playing an important role in industry and research, graphical programming languages as Lab VIEW has a variety of reusable graphical objects applicable in this kind of systems. In this paper we present an effort to represent the knowledge in the development of data acquisition and control systems with graphical programming, modeling the software process by UML notation.

Key-Words: - Graphical Programming, UML, Control Systems, Knowledge Representation

1 Introduction

Data acquisition and control systems with reusable graphical objects are planned to solve complex and ever-changing problems applying knowledge representation in this environment. If the knowledge is captured and reapplied, the systems can become even easier to develop, thus increasing its probability of success. However, even with today's more advanced technology, operational knowledge is never captured, and organizations often spend much time and money rediscovering what they already knew but never took the time to capture. For increasing value and quality, reducing costs and time, the acquisition and knowledge representation are required. UML notation can be used for communicating and representing knowledge. We believe that UML object diagrams can be considered to be a representation of knowledge.

2. Developing knowledge representation

One of the factors that were considered in the development of knowledge representation is the role of system engineering in software engineering, because the software is a part of more complex technological systems. System engineering focuses on a variety of elements, analyzing, designing, and organizing those elements into a system that can be a product or a technology for the transformation of information or control.

2.1 Background

For the knowledge representation of Acquisition and Control Systems with Graphical Programming, was necessary first to develop the software process model for graphical programming presented in figure 1. The graphical programming [1] permits to create functions and even programs represented by graphic objects that encapsulate both data and algorithms. Whether graphical objects are properly implemented, they are reusable across different applications or systems. The model for graphical programming provides the framework, for graphic objects development and incorporates many of the characteristics of the spiral model created originally by Boehm [2]. It is evolutionary and possesses the iterative nature of prototyping model. In this model

the software is developed in a series of incremental releases and consists of three knowledge categories: requirements analysis, management and engineering. In engineering category the term GP refers to graphical programming.

A hierarchical description of a body of knowledge for the software process model is described. The description of the body of knowledge was based in "Software Engineering Body of Knowledge" [3] using too its operational definitions.

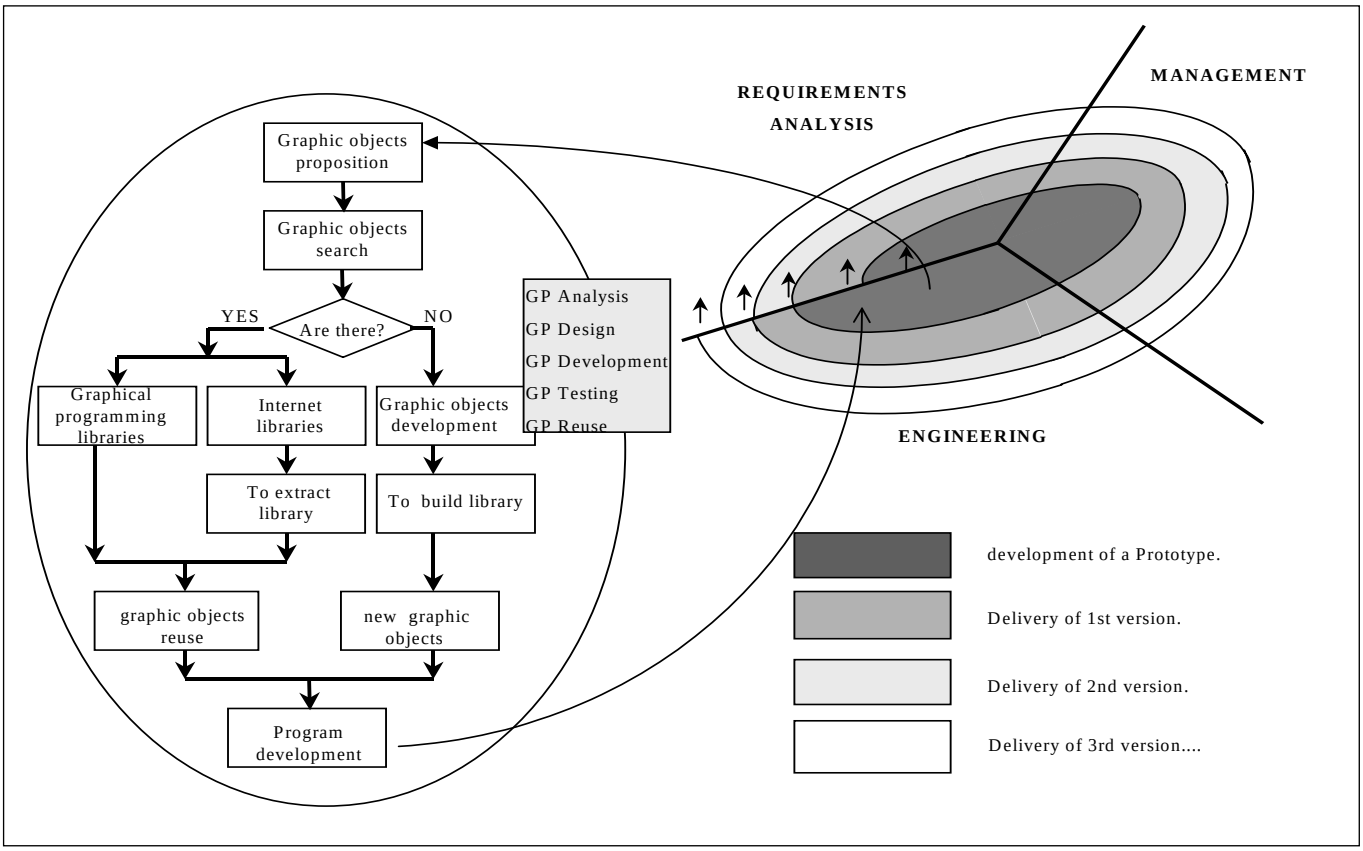

Figure 1. Software process model for graphical programming

In the software process model for graphical programming, the process moves through an evolutionary spiral that starts with analysis of requirements**.** It is here that the problem domain is defined and basic graphic objects are identified. Management establishes a foundation for the project plan**.** The technical work associated with software engineering follows the iterative path shown in the ellipse**.**

2.2 Representing knowledge.

2.2.1 Software engineering body of knowledge.

Knowledge: is used to describe the whole spectrum of content for the model.

Body of Knowledge (BOK): a hierarchical description of software engineering knowledge that organizes and structures the knowledge into three levels: knowledge category KC, knowledge areas KA, and knowledge units KU.

We organize the process model in three knowledge categories: analysis of requirements, management and engineering.

2.1.1.1 Analysis of Requirements. This category establishes a common understanding of the

requirements to be addressed by the software product and include information about the elicitation, analysis and specification.

- **Requirements Elicitation:** it provides knowledge that supports the systematic development of a complete understanding of the problem domain.

- **Requirements Analysis:** it provides knowledge about the modeling of software requirements in the information.

- **Requirements Specification:** it concerns with the representation of software requirements.

2.1.1.2 Management. This category deals with the concepts, methods, and techniques for managing software products and projects and it includes activities concerned with project management, risk management, software quality, and configuration management.

- **Software Project Management:** this area deals with defining project objectives, assessing project needs and resources, and defining the plan for performing the work.

- **Software Risk Management:** this area is concerned with the concepts, methods and techniques that threaten a plan for developing a software product.

- **Software Quality Management:** this area is concerned with the concepts, methods, techniques, procedures and standards for producing high-quality software products.

- **Software Configuration Management:** it deals with the discipline of identifying the configuration of a system a discrete points in time for systematically controlling changes to this configuration and maintaining the integrity and traceability of this configuration throughout the life of the software system.

- **Software Process Management:** this area deals with the management of the technical aspects of the software development.

2.1.1.3 Engineering : this category is concerned with a well-defined and integrated set of activities to produce correct, consistent software products effectively and efficiently and for this model include knowledge about Graphical Programming Analysis GPA, Graphical Programming Reuse GPR, Graphical Programming Design GPD, Graphical Programming Construction GPC, and Graphical Programming Testing GPT.

- **Graphical Programming Analysis:** this area is concerned with the requirements addressed by the software product and the development of a series of models that describe computer software and satisfy a set of defined requirements.
- **Graphical Programming Design:** this area is concerned with the transformation of the statement of requirements into a description of how these requirements are to be implemented.

- **Graphical Programming Development:** this area is concerned with knowledge about the development of the software components that are identified and described in the design documents.

- **Graphical Programming Testing:** this area is concerned with establishing that a correct solution to a problem, embodied in the statement of the requirements, has been developed.

- **Graphical Programming Reuse**: this area is concerned with the reuse and creation of reusable components.

2.1.2 Representing knowledge of Acquisition and Control Systems with Graphical Programming using UML.

The UML notation permits to organize and to class knowledge, such knowledge is captured in a model consisting of various modeling elements, and it is represented with distinct sets of diagrams. The model

itself captures the knowledge, and the diagrams represent the knowledge in a communicable form. The UML notation [4, 5] offers diagrams through which we represent the knowledge of Acquisition and Control Systems with Graphical Programming. The UML gives a common vocabulary to deal with software problems, it begins with the construction of a model. A model is an abstraction of the underlying problem. The domain is the actual world from which the problem comes. Models consist of objects that interact by sending messages. Objects have (attributes) and things they can do (behaviours or operations). The values of an object's attributes determine its state.

2.1.2.1 Analysis of Requirements. For representation knowledge of requirements we use the next kinds of UML modeling diagrams:

- **Use case diagrams:** They describe what a system does from the point of view of an external observer. The emphasis is on *what* a system does rather than *how*. A **use case diagram** is a collection of actors, use cases, and their communications.
- **Sequence diagrams:** They are interaction diagrams that detail how operations are carried out, what messages are sent and when. Sequence diagrams are organized according to time.
- **Protocols**: They are used for modeling behaviour; a protocol is a specification of desired behaviour, an explicit specification of the contractual agreement between the participants in the protocol.
- **State machines:** The specification of valid protocols sequences is done using standard UML state machines

Figure 2 shows the use case diagram, that represents the knowledge acquired in the analysis of requirements, and a use case description is presented.

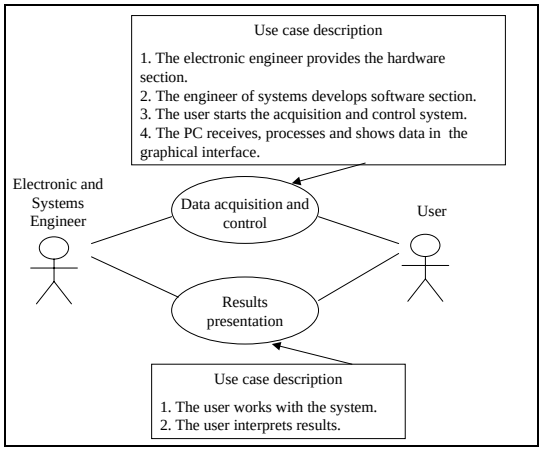

Figure 2. Use case diagram for knowledge representation in the analysis or requirements.

Figures 3 and 4 shows the interaction between actors and scenario about operation details.

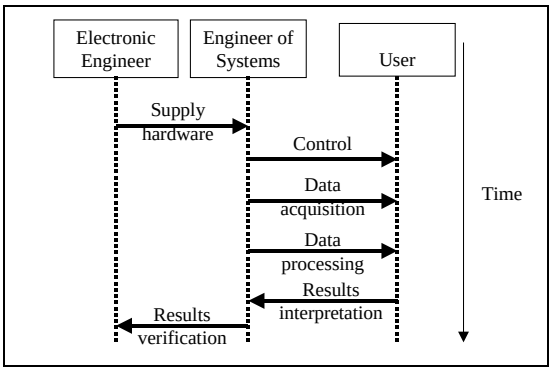

Figure 3. Sequence diagram about personal interaction within data acquisition and control system.

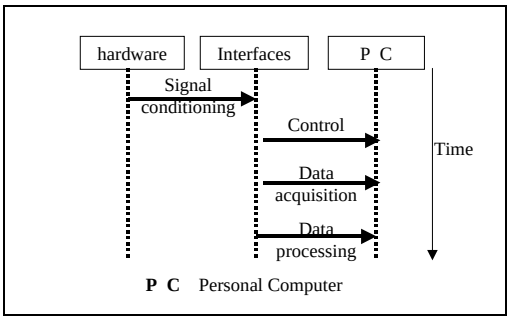

Figure 4. Sequence diagram showing the hardware interaction within data acquisition and control system.

Figure 5 is a protocol [6] that shows a specification of desired behaviour of a data acquisition and control system; here the incoming and outgoing signals are exhibited.

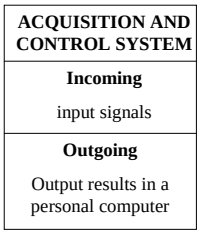

Figure 5. Protocol for behaviour of a data acquisition and control system.

Consider the abstract machine in figure 6, it represents the most abstract level of behaviour a data acquisition and control system and shows how a simple state machine represents the behaviour of a real-time system.

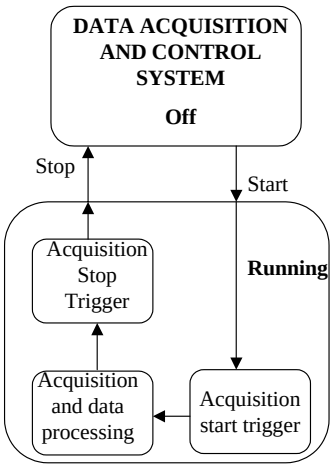

Figure 6. Protocol for behaviour of a data acquisition and control system.

2.1.2.3 Management: In this category we represent the knowledge acquired with a use case and it represents concepts, methods, techniques, and standards required in project, risk, quality, configuration and process management. The software project management is an umbrella activity within software engineering, and stars after requirements analysis and continues throughout the definition, developments, and support of acquisition and control system. Risks management is a series of steps that help a software team to understand and manage uncertainty. Software quality management encompasses procedures for the effective application of methods and tools, formal technical reviews, testing strategies, procedures for change control, and procedures for assuring compliance to standards and measurement and reporting mechanisms. Software configuration management identifies, controls, audits, and reports modifications that invariably occur while software is being developed and after is has been release to a customer. Software process management enables manager to improve and apply a software process. Figure 7 shows the use case diagram that represents the knowledge acquired in the management category.

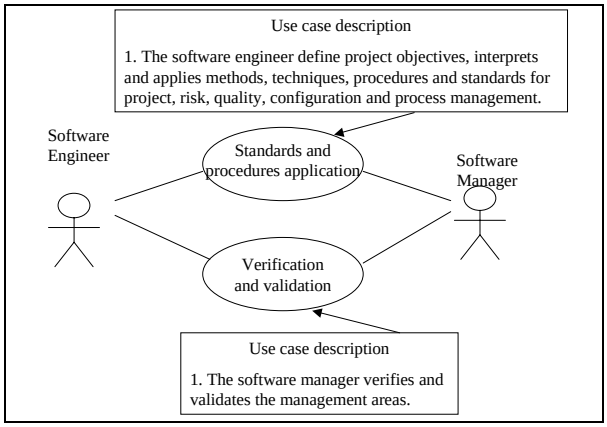

Figure 7. Use case diagram for knowledge representation in the management category.

2.1.2.4 Engineering. In engineering category we have considered five knowledge areas: graphical programming analysis, graphical programming design, graphical programming development, graphical programming testing, and graphical programming reuse. For knowledge representation of engineering category [7] we apply use case that is presented in figure 8.

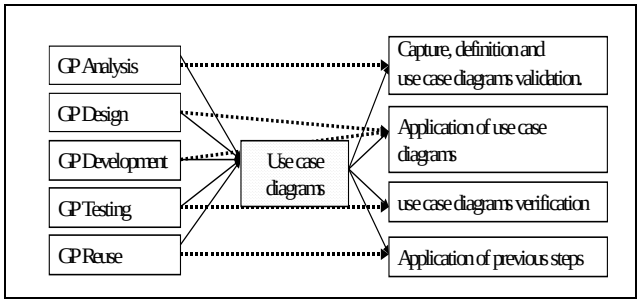

Figure 8. Knowledge representation of engineering category.

Because the graphical programming language LabVIEW produces objects, besides of the use case diagrams, we are proposed to use an activity diagram for knowledge representation of the design and graphical programming development, it models graphical objects and it is show in figure 9.

increases its probability of success avoiding to spent much time and money. It is highly advantageous for software developers and software managers to represent knowledge, for increasing quality, and for reducing costs.

References

1. G. W. Johnson, LabVIEW Graphical Programming, Practical applications in Instrumentation and Control, McGraw-Hill, 1994.
2. R. Pressman, Software Engineering a Practitioners Approach, McGraw-hill International Edition, 2001.
3. T. B. Hilburn, I. Hirmanpour, S. Khajenoori, R. Turner, A. Qasem, A Software Engineering Body of Knowledge, Version 1.0, Technical Report, CMU/SEI-99-TR-004.

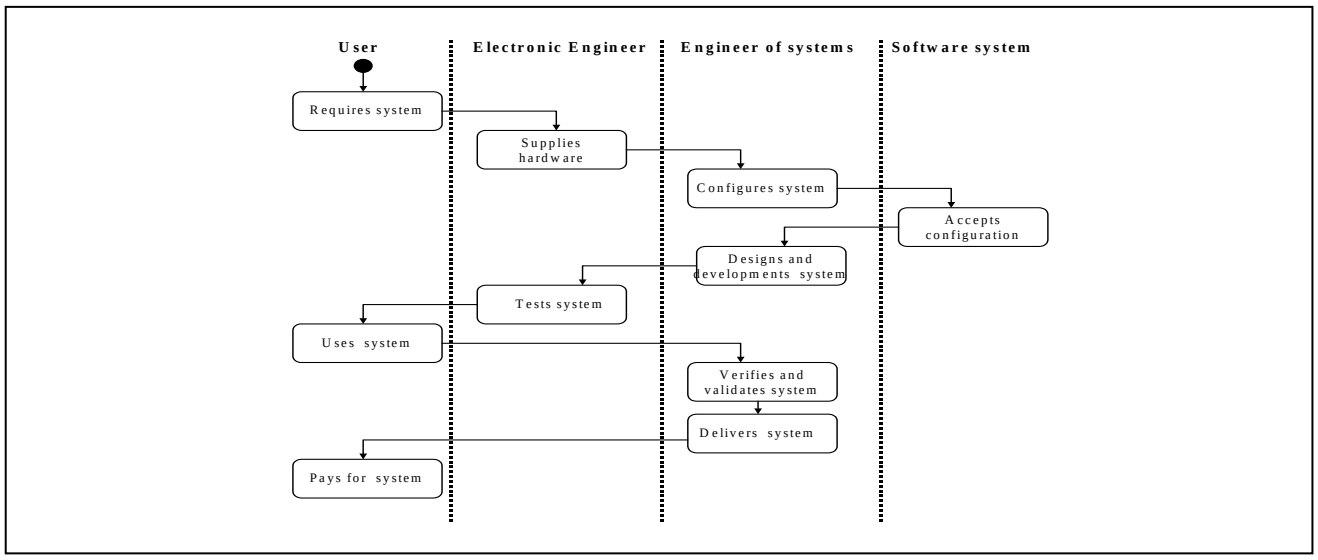

Figure 9. Activity diagram for knowledge representation in the management category.

3. Conclusions

Knowledge representation with UML notation leads to acquire and represents the software engineering knowledge in data acquisition and control systems. This helps to solve problems through the application of knowledge. The UML diagrams are a good option to capture lessons learned, good practices in software engineering, and reapplying knowledge the systems

4. Borland, Practical UML A Hands-On Introduction for Developers.
5. Unified Modeling Language, http://www.rational.com/uml/
6. B. Selic, J. Rumbaugh, “Using UML for Modeling Complex Real- Time Systems” http://www.rational.com/products/whitepapers/
7. P. Letelier, Desarrollo de Software orientado a objetos usando UML, http://www.dsic.upv.es/~uml/.