5-th International Conference on System Simulation & Scientific Computing
Shanghai, China, Nov 3-6, 2002, paper 04F008

UML SUPPORTED DESIGN OF MECHATRONIC SYSTEM

Zbigniew Mrozek[1], Tao Wang [2] and Minrui Fei [2]
(1) Cracow University of Technology, PL 31-155, KRAKÓW, Poland
(2) Shanghai University,Post Code , 200072
Shanghai,China

ABSTRACT

Information transfer plays an important role in operation of mechatronic system. This can be easily presented on UML (Unified Modelling Language) diagrams. Author believes that terminology and notation of visual modelling with UML can be adopted as common language for design of the mechatronic systems and as documentation tool on every design phase.

INTRODUCTION

UML was introduced as language for modeling of the information systems, but it can be used to describe all elements of mechatronic system.

UML LANGUAGE

UML is derived from three other products: OMT (object modelling technique) by James Rumbaugh, Booch method by Grady Booch and OOSE (object oriented software engineering) by Ivar Jacobson. The UML language was improved many times until version UML 1.3 was accepted as proposal for standard in year 1999. The current UML version specification is 1.4 and proposals for version 2.0 are under discussion. UML represents a collection of the best engineering practices that have proven successful in the modelling of large and complex systems. Many successful attempts have been considered to extend the application of UML to areas beyond informatics.

Any complex system can be presented by a set of nearly independent views of a model. Single view is not sufficient. *Use case* diagram and *class* diagram (there are described later) are probably used in all UML supported projects. The choice of what other diagrams are created depend on how a problem is attacked(Figure 1). In addition to

- *use case diagram*
- *class diagram*

one can create any of behavioural diagrams:

- *statechart* diagram
- *activity* diagram
- interaction diagrams*: sequence* and *collaboration*

and implementation diagrams:

- *component* diagram
- *deployment* diagram

Some CASE tools supporting UML design may offer their own extensions and diagrams which are not included in current UML specification (e.g. *system architecture diagram* in Real Time Studio from Artisan Software Tools),

There is no need to use all existing types of behavioral diagrams in applications. That is, it is sufficient to have a deep knowledge of a small subject of carefully chosen diagrams. As Brugge [3] points out; "You can model 80% of most problems by using about 20% UML". It means that a UML module should present only a part of carefully selected elements of the UML language. In the author's opinion, use case diagrams, scenarios, class and object diagrams, two or three behavioural diagrams (e.g. sequence and state),

USE CASES

The main tool used for requirements elicitation during analysis phase are use cases and scenarios. Use case is represented by an ellipse on use case diagram (figure 2). It captures sub-system functionality as seen from the point of view of end user or domain expert. It helps to understand how the system should work. This is an important job, as original problem description may be incomplete and some requirements may conflict with others. Requirements are imperfect and not always understood. Once requirements are really understood, the client may realizes they are wrong. Blaming the client for a defective problem statement is not an acceptable option.

Collection of use cases describes different behaviour of the system and shows how it interacts with external actors.

Actor is a human user, another system, sensor or anything located outside of the actual system that will initiates a use case and will interact with the system. Actor is depicted as a simple icon of a man. Defining actors is essential to set the border between the system under development and its external environment.

An example of use case diagram for Robocop football player robot is given below.

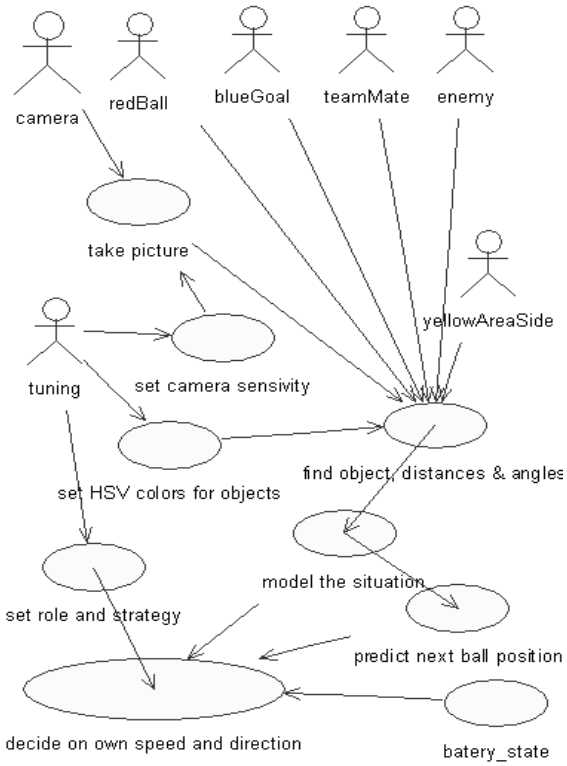

Figure 1 Use case diagram for football player robot (robocop)

SCENARIO

Scenario is defined as set of messages (or description) in natural language, describing chosen sequence of actor and system interactions. It describes details of use case functionality.

CLASS AND OBJECT DIAGRAM.

When use cases or scenarios are analysed, object activities are described although classes are not defined at all. This may be very strange to students with some background knowledge in C++ or Java languages and who, in general, accept that an object is an instance of class. Here in visual programming using UML a class may be defined and drawn later, as generalization of chosen similar objects. A class diagram may contain classes and objects. If there is no other class in such a diagram, the diagram is named an object diagram.

There is no officially recognized methodology to identify object and classes. Therefore, different programmers may define a different set of classes and objects for the same problem.

It is not a good idea to design a complete class or object diagram when scenarios and use cases are ready. This is because some changes in object hierarchy, naming, methods and attributes may be inevitable, when other diagrams are designed. This is true especially when sequences or state diagrams are under preparation. In general, when a good case tool supporting UML programming is used, the class diagram is corrected automatically when objects or class names, types, methods, attributes or parameters are changed in other diagrams.

BEHAVIORAL DIAGRAMS

There is no need to use all existing types of behavioural diagrams in applications. That is, it is sufficient to have a deep knowledge of a small subset of carefully chosen diagrams. As Brugge [3] points out; “You can model 80% of most problems by using about 20% UML”. It means that a UML module should present only a part of carefully selected elements of the UML language. In the author’s opinion, use case diagrams, scenarios, class and object diagrams, two or three behavioural diagrams (e.g. sequence and state), component and deployment diagrams, OCL (Object Constraint Language) should be considered when designing a UML module.

SEQUENCE DIAGRAM

Sequence diagram is graphical model of scenario with emphasis on interaction between actor and system and on communication between objects inside the system. The diagram shows what objects do to implement a scenario. During preparation of diagrams one can verify requirements specification and scenarios against omissions and inconsistencies. Similarly, one can verify existing classes and objects. Missing classes and objects should be defined at this stage. Methods and attributes defined here should be identified automatically by the

CASE package and used immediately to update existing classes and objects.

STATE DIAGRAM

If an object's behavior is more complicated, a sequence diagram is not sufficient and a state-chart diagram should be designed. Transition can be event triggered or time triggered

ACTIVITY DIAGRAM

Represents data and/or control flow as flowchart. It can be multi-threaded to initiate parallel paths. Good way to look at system-level behaviour and emphasize concurrent operations.

CASE TOOLS

When building diagrams, one will verify if already defined objects are useful for tasks and services described with the new state or sequence diagrams. He has to decide what methods (its name, type and parameters) and attributes (its name and type) are needed. Most CASE tools (Rational; Rose [8] and Real Time Studio [9] were used by authors) have a database with data of objects defined within a project. In case of any inconsistency, user is immediately alerted to correct the error. This speeds up design and helps to avoid many errors

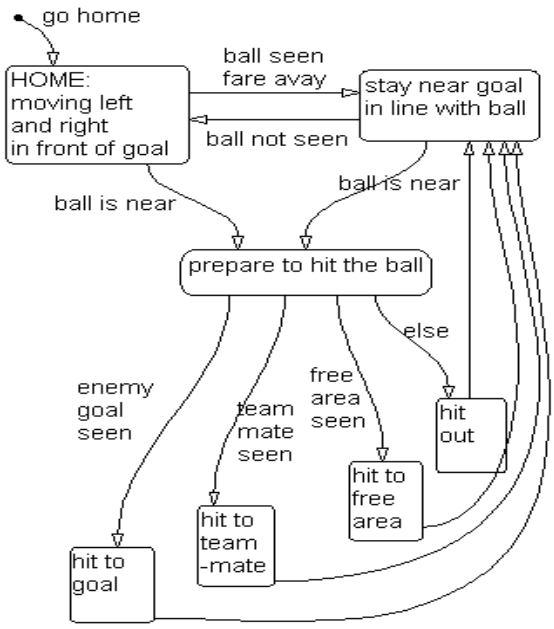

Figure 2 State diagram for goal keeper robot

BRAINSTORMING

Brainstorming is used to identify possible solutions to problems and potential opportunities for improvements. This technique is used for tapping creative thinking of a team to generate and clarify a list of ideas, problems and issues. We use brainstorming to find all use cases and actors necessary to build a use case diagram. Brainstorming is useful to identify states and transitions for state diagram. There are two phases in the brainstorming procedure.

During the **generation phase** the purpose (target) of the brainstorming session is clearly stated, each team member takes a turn in a sequence, stating a single idea where possible, new ideas are build on others ideas, all ideas are recorded and should be seen by all the participants (using whiteboard or overhead is recommended). At this stage, ideas are neither criticized nor discussed. the process continues until no more ideas are generated.

During the **clarification phase** all ideas are reviewed to make sure that each person understands them all. Evaluation of ideas will occur after the brainstorm session is completed.

CONCLUSIONS

The UML has the advantage that it reveals gaps and inconsistencies in the requirement's specification at earlier stages of design, as well as making it easy to understand and modify visual modelling diagrams. Unification and precision of notation is important for large and interdisciplinary projects. User may transfer already defined classes and other elements between different diagrams and projects to reuse them. This accelerates work on the project.

Using commercially available specialised CAD tools and CASE packages, visual UML programming may greatly improve productivity of a software design team by cutting down development time and improving final product quality (in accordance with ISO 9000 standards).

ACKNOWLEDGEMENTS:

Authors wish to express their gratitude to ARTiSAN Software Tools, Inc (GB); Rational Software Corporation (USA) and Premium Technology Sp zoo (Poland) for free evaluation license for following

software: Real-time Studio, Rational Rose Suite and Rational Rose RT.

REFERENCES

(1) Booch, G. Rumbaugh, J. Jacobson I The Unified Modelling Language User Guide, Addison Wesley, 1999

(2) Bruegge B, Dutoit A, Object-Oriented Software Engineering: Conquering Complex and Changing Systems, Prentice-Hall, 1999.

(3) Kobryn Cris, UML 2001 A Standardization Odyssey, Communications of the ACM Vol.42,No.10, pp.28-37, October 1999.

(4) Mrozek Z, UML as integration tool for design of the mechatronic system, in Second Workshop on Robot Motion and Control, pp 189-194, ed. Kozlowski K, Galicki M, Tchoń K, Oct 18-20, 2001, Bukowy Dworek, Poland.

(5) Mrozek B,. Mrozek Z, MATLAB 6, poradnik użytkownika, ISBN 83-7101-449-X, PLJ Warszawa 2001.

(6) Mrozek Z, Methodology of using UML in mechatronic design (in Polish), pp.25-28, Pomiary Automatyka Kontrola 1, 2002.

(7) OMG Unified Modeling Language Specification (draft), Version 1.4, February 2001. and other OMG (Object Management Group) standards, http://www.omg.org:

(8) Rational Rose Suite, Rational Rose RT and other software from Rational Software Corporation.

(9) Real-time Studio, ARTiSAN Software Tools, Inc. July 2001.

(10) Uhl T, Bojko T, Mrozek Z, Szwabowski W, Rapid prototyping of mechatronic systems, Journal of Theoretical and Applied Mechanics, pp.655-668, vol.38.3, 2000

(11) Uhl T, Mrozek Z, Petko M Rapid control prototyping for flexible arm, Preprints of 1-st IFAC Conference on Mechatronic Systems, vol II, pp 489-494, Darmstadt, September 18-20, 2000.

(12) Wei-Min Shen et al, Building Integrated Mobile Robots for Soccer Competition, Proc. IEEE Int. Conf. on Robotics& Automation, pp.2615-2618, Leuven, Belgium May 1998