Application of Object Oriented Domain Analysis and Design in the Model Driven Architecture Framework

Krishna Mohan Tedla

Dr. Emanuel S. Grant, PhD

Computer Science Department
University of North Dakota
Krishna@cs.und.edu

Abstract

The use of software development technologies is expanding rapidly in domains that use software applications and those in which the technologies are being applied for the first time. As the technologies change the software developers are unsure which of the available technology should be selected. Software modeling involves the designing of software applications before coding is done. The Object Management Group's Model Driven Architecture is based on the development of Platform Independent Models and Platform Specific Models. It incorporates the Unified Modeling Language, a standard software modeling language to develop models. The goal of this research work is to demonstrate the benefits of using the Model Driven Architecture towards producing applications that possess a high degree of scalability, reusability, robustness, and maintainability. This work focuses on the production of domain analysis and design models for a specified non-trivial problem domain.

Introduction

The Object Management Group (OMG) is a non-profit organization established in 1989. With its mission to help computer users solve integration problems by supplying open, vendor-neutral interoperability specifications [1]. The organization functions include the establishment of industry guidelines and detailed specifications to provide a common framework for application development. The conformance to these specifications makes it easy to develop heterogeneous computing environments across major hardware platforms and operating systems.

OMG defines software development as including the models of real world through representation of objects. Objects are the encapsulation of attributes, relations and methods and are identified as program components. A key advantage of using object-oriented system is its ability to expand functionality by extending existing components and adding new objects to the system. Object-oriented development results in faster application development, easy maintenance, scalability, and reusability.

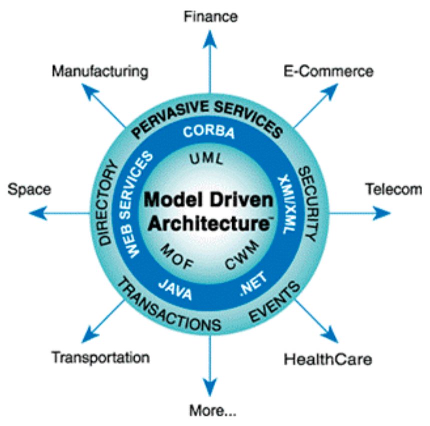

Figure 1: OMG's Model Driven Architecture [2]

Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) defines an approach to system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform [1]. The MDA defines an architecture for models that provides a set of guidelines for structuring specifications which are expressed as models.

The Model Driven Architecture [2]:

- Integrates what has been built, with what is being built and what will be built in the future
- Remains flexible in the face of constantly changing infrastructure
- Lengthens the lifetime of software, and lowers maintenance costs.

The Core

The core of the MDA is at the center of Figure 1 which consists of the OMG's modeling standards: UML, MOF and CWM.

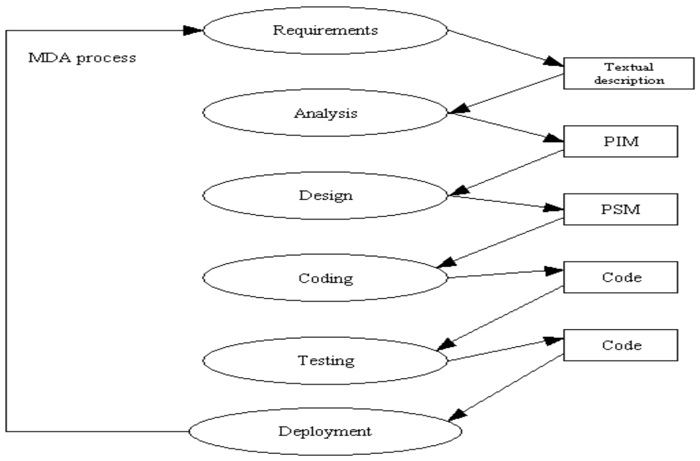

Figure 2: MDA software development life cycle [3]

MDA development Life Cycle

Figure 2 above depicts the MDA software development life cycle. In the MDA life cycle the textual description of the problem to be solved and end user interest of needs are collected during the requirements phase. The textual description collected during requirements phase is used as input to the analysis phase. During analysis phase the collected textual description requirements and the Platform Independent Model (PIM) are developed. The PIM's developed are used as input to the design phase and using different mappings techniques discussed below the PIM's are transformed into one or more Platform Specific Models (PSM's). The PSM's developed are input to the coding phase. In the coding phase, code generation tools are used which take PSM's as input and produce executable code.

The MDA software development life cycle is similar to traditional software development life cycle. The major difference between the MDA development and a traditional one is that in the MDA development life cycle formal models are created using UML. The MDA specifies three types of models in the development life cycle, namely they are PIM, PSM and executable code.

Platform Independent Model

The PIM is the first model defined in the MDA framework. PIM's are used to present a high level abstraction through to detail design models of the application domain that is independent of any specific implementation technology. The description of PIM's is an

iterative process, so multiple levels of PIM’s are defined. The base PIM’s express only business functionality and behavior of the system without any technology details. The models defined from base PIM’s using iteration include some aspects of technology even though platform specific details are absent.
The Unified Modeling Language (UML), a widely accepted language modeling whose concepts are used to define PIM’s, which enhances the power of MDA. The UML diagrams such as class diagrams use case diagrams, activity diagrams, sequence diagrams, collaboration diagrams, and state diagrams are used to define PIM’s.

Platform Specific Model

A PIM is transformed into one or more platform specific models using mapping techniques. A PSM specifies software system in terms of implementation constructs available in a specific technology platform. UML extensions are used to introduce new notations and terminology to represent and use in the models. A UML profile is a set of stereotypes, tagged values, constraints and notations that are used to represent UML models in a specific domain. Stereotypes are used to derive and create new kinds of model elements using existing ones. Tagged values are used to create new information specific to model elements. Constraints are used to create conditions as Boolean expressions that can be applied to model elements. UML extensions and specializations provide power to transform a PIM to into one or more PSM‘s. UML profiles are used for transforming/mapping from PIM to PSM. A UML profile is a standard set of extensions which consists of tagged values, stereotypes defines a UML environment for modeling to a specific platform.

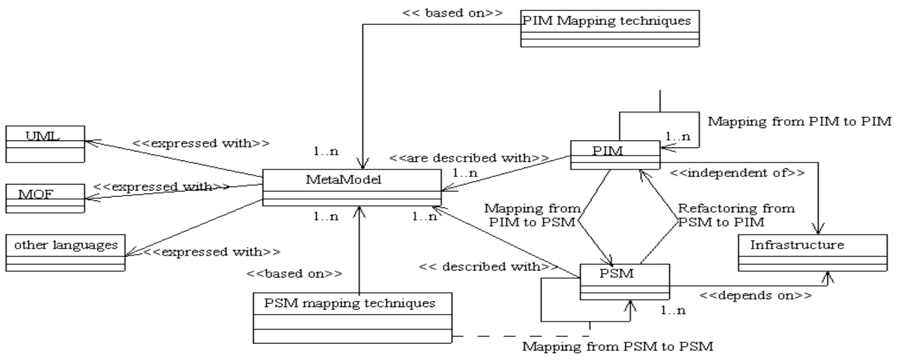

Figure 3: MDA Metamodel Description [1].

The Mappings of Figure 3 are sets of rules and techniques used to modify one model to obtain another. Figure 3 shows the MDA Metamodel description which illustrates various mappings. Mappings are used for transforming of models from:

PIM to PIM: transformations are related to model refinement. PIM to PIM mapping is an iterative process independent of platform details. In every iteration the generated output model contains more details about the problem domain than the one in the previous iteration. For example some details are abstracted in the analysis model, but are elaborated in the design model. An example mapping is transformation from an analysis to design models.

PIM to PSM: PIM to PSM mapping is an iterative process dependent on platform specific details. This transformation is used when the PIM is refined enough with complete details and has to be projected to some specific technology platform. A highly detailed PIM with complete information about the application can be transformed in to PSM. An example mapping is transforming from a logical model to some specific platform like CORBA.

The ways to move from a PIM to PSM are:

- The transformation is performed manually by hand without referring to established patterns and procedures.
- The transformation is performed manually using established patterns to convert from the PIM to a particular PSM.
- An algorithm defined by established patterns is implemented in a MDA tool which produces a skeleton PSM, which can be completed by hand.
- Applying an algorithm within a MDA tool produces the entire PSM.

Code

The final step in the MDA development is the transformation from PSM to executable code.

Benefits of MDA

- Separation of PIM's from PSM's allows reacting quickly to changing functional and technological platform requirements.
- Extends the life of the software system
- Modeling techniques separate a system into cohesive components, which simplifies each system component and makes it easier to create, develop, reuse, maintain and improve developers' productivity.
- PIM's can be reused to produce multiple component implementations in different platforms.
- As the code is generated from models, it lowers the maintenance costs for the software system.
- PIM models helps to improve communication between team members and early elimination of defects in the system.
- Reduces cost throughout the application development life cycle.
- Increases the return on investments.
- Includes emerging new technology benefits in to existing systems.
- MDA transformations are executed by tools rather by hand. An important aspect of MDA is the automated transformations from PIM to PSM.

MDA in Research work

In my research work of developing Executable UML models for the problem domain I am using the MDA framework to generate the models. By getting the specifications for a problem domain I will develop PIM's using UML modeling concepts. The high level PIM design models developed can be transformed to PSM's and then to executable code for any specific platforms using transformation tools of PIM's to PSM's and PSM's to code. As the transformation tools are highly expense and no free tools are available, my work ends with developing detail level design PIM models

The Unified Modeling Language

The Unified Modeling Language (UML) is a modeling language for specifying, constructing, visualizing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems [4]. The UML plays an important role in the developing of object oriented software projects and processes. The UML uses graphical notations to express the design of software projects, which helps project teams to communicate, explore designs, and validate architectural design of the software.

Why we need Modeling

In recent times software systems are becoming more complex and large making it difficult to comprehend the system in it's entirely. We need models to build the complex systems. Modeling is the designing of software applications in support of coding. Modeling is a means to visualize design and checks it against requirements before the start of coding. Developing a model for a software system prior to its coding is as essential as having a blueprint for constructing a building. As complexity of systems increases need for modeling and visualization becomes essential and important. Good models assure architectural soundness and communication among different project teams. By using models for software projects the developers can assure that business functionality is complete and correct, end-user requirements are met and design supports requirements for scalability, robustness, security and extendibility and many other characteristics.

A modeling language is a collection of graphical notations, which are used to express analysis and design parts of a problem domain in visual representation.

A modeling language must include [4]:

- Model elements --- fundamental modeling concepts and semantics
- Notation --- visual rendering of model systems
- Guidelines --- idioms of usage within the system

Why the UML

As the strategic value of software increases for many companies the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time to market. These techniques include component technology, visual programming, patterns and frameworks [4]. Businesses need techniques to manage the complexity of systems as scope and scale of systems increase. The modeling language should solve recurring architectural problems like physical distribution, concurrency, replication, security, and load balancing and fault tolerance. The UML is designed to meet all the needs in the industry. The key motivation for UML developers is to create a set of semantics and notation that adequately addresses all scales of architectural complexity across all domains [4].

Goals of the UML

The primary goals of the UML are [4]:

- Provide users an easy to use, highly expressive visual modeling language to develop meaningful models.
- Furnish extensibility and specializations mechanisms to extend the UML core concepts.
- Support specifications that are platform independent of any programming languages and technologies.
- Provide a formal view to understand the modeling languages.
- To increase the growth of the object oriented tools market.
- Support high level development concepts like components, collaborations, frameworks and patterns.
- Integrate best development practices in the software industry.

Features of the UML

The goals of the unification effort is to keep the modeling language simple, to remove elements of existing Booch, OMT and OOSE that did not work and to add elements from that are effective. Some elements were added to UML as they have proven useful in practice in other modeling languages.

The new concepts that are in included in UML are [4]:

- extensibility mechanisms (stereotypes, tagged values and constraints),
- threads and processes,
- distribution and concurrency,
- patterns/collaborations,
- activity diagrams (for business process modeling),
- refinement (to handle relationships between levels of abstraction),
- interfaces and components,
- A constraint language.

Class diagrams

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among objects [5]. A class icon of class diagram is depicted in figure 2 below. A class is a collection of group of things that have similar attributes and common behavior. Classes are composed with three parts: a class name, attributes and operations. An attribute is a property of a class. An operation is a task that a class can do or another class can do to a class. A class diagram represents the static behavior of the system.

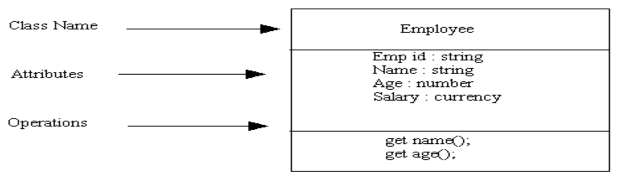

Figure 4: Class icon in a class diagram

Class relationships: The relationships among different classes are

Association – when classes are connected conceptually the connection is called an association.

Multiplicity – when a number of objects from one class relate with a single object of another class, the relationship is called multiplicity.

Aggregation – is a relationship in which a class consists of a number of component classes. Aggregation is also a called a part-whole association.

The class relationships association, multiplicity, reflexive, dependency, aggregation, composites between different classes is depicted in figure 3 below.

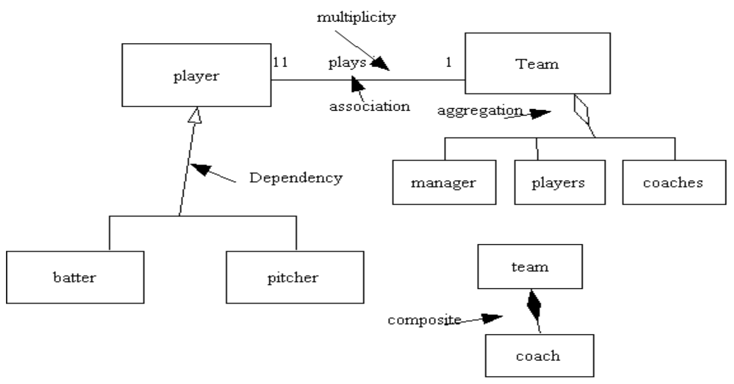

Figure 5: Class relationships in a class diagram

Domain Analysis

Domain analysis is the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and development histories, knowledge captured from domain experts, existing theory and emerging technologies within a domain [6]. Domain analysis deals with the identification, acquisition and representation of software requirements specification and software implementation knowledge for real world problem domains. Reusability is using the software components produced specifically to be reused in more than one application. The purpose of domain analysis is to support reusability of software components and help reuse community.

Reuse based software development

The conventional software development becomes more powerful when it adopts reusability of software components and techniques. The application of reusability depends on proper documentation of previous problem descriptions and methods to derive solutions to problems. During software development it is very difficult to identify the right components that can be reused for a project. This difficulty of finding the right components to reuse raised the necessity of reuse infrastructure. The reuse infrastructure is the database of various repositories of domains items that can be reused. Each repository contains reuse items of one domain that can be reused within the various software application projects of that particular domain. The software development organizations can refer to the reuse infrastructure to identify the components that can be reused with in their software projects.

Problem domain - refers to a class of problems that belong to the same category and deals with similarly related information.

Information belongs to a problem domain if [7]

- strong relationships exist among the objects of information with respect to some class of problems,
- software development community exists that has interest in solving the problems of a domain,
- the software development community wants to have software solutions to the problems of a domain and
- the software development community has access to knowledge base, which can be applied to solve the problems of a domain.

Domain model – the purpose of domain analysis is to produce a model of the problem domain which is referred as domain model. A domain model is a model of the problem domain whose elements are classes and the relationships between the classes. A domain model includes the complete details about the problem for specifying the model, the rules for transforming specifications into code.

The domain model serves as [7]

- a reliable unified source of reference when ambiguities come during the analysis of problems or later during the implementation of reusable components in the application,
- a repository of the shared knowledge base for teaching and communication and
- a specification to the implementer of reusable components.

Domain engineering

Domain engineering is an activity for building reusable components [6]. Domain engineering includes all the activities of domain analysis, domain design and domain implementation for building software core assets. Domain design is the process of developing design models from domain analysis products and the knowledge base of reuse infrastructure. Domain engineering focuses on a set of related systems with in a domain. Domain engineering process starts with domain knowledge, business objectives of a development organization and creates customized solutions to each set of problem with in a domain [8].

Domain analysis process

Figure 1 shown below summarizes the inputs, outputs and agents that support the domain analysis process.

Inputs

The knowledge about a problem domain and how to implement the solutions to problems in the domain can be acquired from different sources. The inputs to the domain analysis process can be obtained from the following.

- The technical literatures such as textbooks, journals and manuals.
- The already existed applications which can be used for investigating as source code, design documentation, and user manuals for implementation.
- customer surveys and market analysis.
- advices and suggestions from human experts in the problem domain.
- Historical records of evolution in the domain.
- Requirements of legacy, current, and future systems.

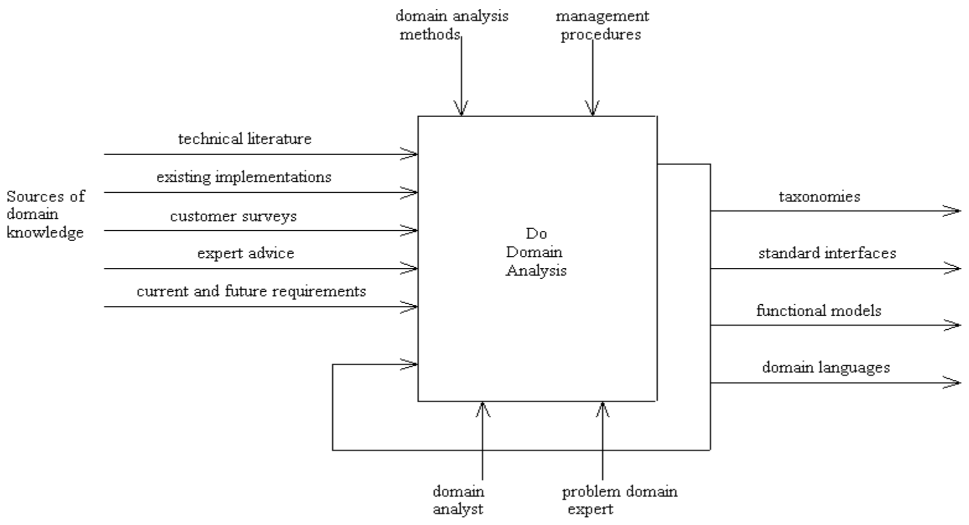

Figure 6: Context view of Domain Analysis [6]

Outputs

The output of the domain analysis process is domain model. The model should contain at least [7]

- a definition of the concepts used in the specification of software problems and software systems,
- a definition that constitutes major software design decisions, trade offs, and justifications,
- Software implementation plans i.e. how to get the code from the specification.

Mechanisms

A number of tools are useful for developing domain models. The list of tools include editors, analyzers for the modeling languages, data and knowledge base browsers, viewing tools, dictionary management, reverse engineering tools, tools for maintaining, indexing and retrieving documentation.

Actors

A domain expert's functions in the domain analysis process are [7]

- to help identify a domain for the specification and implementation of the system,
- to disambiguate the vocabulary/terminology in problem domain so that the semantics of application-specific domain concepts are refined.
- to provide justifications for specifications and implementation concepts used in the problem domain, and
- to act as reviewer and critic/analyze the resulting domain models.

Domain analyst's functions in the domain analysis process are [7]

- to conduct knowledge acquisition,
- to organize and format the information obtained about the problem into a domain model,
- to verify the correctness of the domain models produced,
- to validate the information in the domain model via existing systems and
- to analyze the effect of changes on the model and evolve the model.

Development Methodology

Object-oriented analysis and design of a domain is the consideration a problem domain and solution to the problem from the objects point of view. During object-oriented analysis phase we study the problem domain and obtain requirements of the domain and express the requirements in terms of objects of the domain. During object-oriented design phase we express a solution to the problem domain with objects identified from the analysis phase. In the object oriented design phase a solution is obtained by identifying the objects with their attributes and operations and by identifying how the objects interact with each other by passing messages.

In my research the problem domain investigated is that of a ball game domain which includes baseball, softball, and the English game of cricket. In this problem domain I will be applying object oriented analysis and design in the Model Driven Architecture (MDA) framework and generate Platform Independent Models (PIM's) using modeling language UML.

The methodology for the research is to read and understand the game playing rules of the three ball games and identify and note the static entities and dynamic entities of each game. After analyzing the playing rules of all three games I will identify the commonalities of rules in the three games and develop models of the games. For example some commonalities of the three games are; in baseball and softball the pitcher pitching a ball to the batter is similar to a bowler bowling a ball to a batsman in cricket; runs or score by the batter in softball and baseball is similar to the runs by batsman in cricket; the batter/batsman getting out in the game, the umpires are on the game field, in the three games, etc.

The static entities are used to develop UML class diagrams and dynamic entities are used to develop the use case diagrams, activity diagrams, and collaboration/sequence diagrams using the UML notation. These PIM models developed for the domain can be used to develop Platform Specific Models (PSM's) in specific platform and programming language. PSM's generated can be input to code generator tools and the code in specific platform can be generated. The PIM's developed can be used by the application developers to develop game training applications, game simulation and coaching applications without the burden of reading the rules and obtaining the requirements for their applications i.e. the requirements analysis phase becomes easier as all the requirements of the domain are formulated in terms of models. Therefore the PIM's developed can be used by developers for any number of software applications to be developed in this problem domain.

The UML class diagrams for cricket, baseball and softball are shown in Figure 7 and Figure 8 below. The class diagram models static entities in the games. In Figure 7 i.e. game of cricket the various classes identified are game of cricket which is played by the two teams and a result is achieved at the end of the game. The players on the field and their field positions. The players can be batsman, bowler and fielders. Batsman instantiates runner to make runs. Wicketkeeper is a fielder who stands behind the wickets at the batsman side. Cricket has two breaks which are drink break and lunch break. The game is supervised by the three umpires whose names are main umpire, leg umpire and TV umpire.

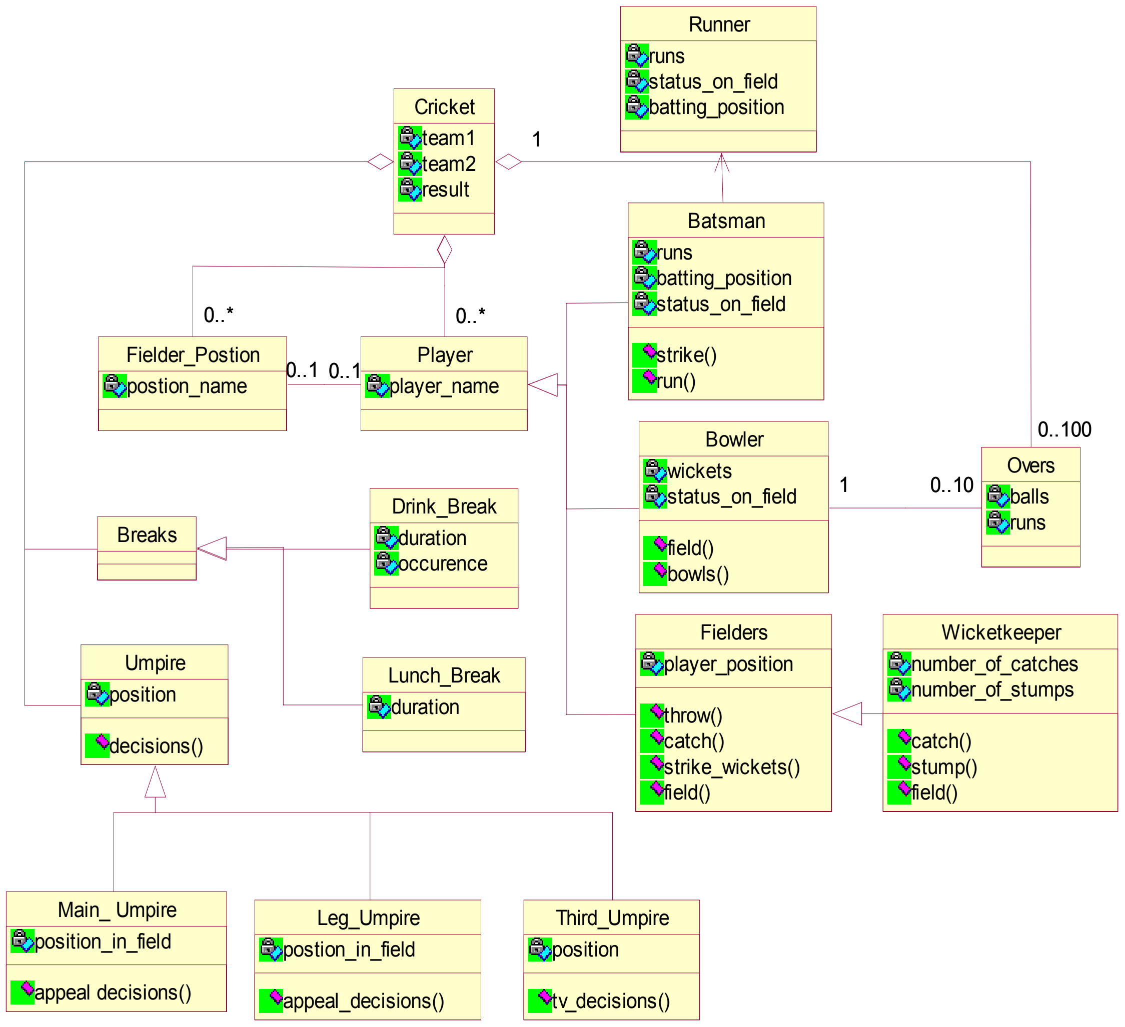

Figure 7: Class diagram for Cricket

Figure 8 below depicts the class diagram of baseball and softball games. The static entities and rules of the baseball and softball games are similar with minor differences and by type of pitching. The game is played by two teams and a result is achieved at the end of the game. It has the players on the field and their field positions. The players are batter, pitcher and fielders. Batter instantiates zero or more base runners by playing the game. Pitcher pitches the ball to the batter. Catcher is a special type of fielder standing behind the batter at the home plate. Each game is played in nine innings or more to achieve a result. Two base coaches stand on first and third bases go guide the runners. Game has four umpires with one on each base.

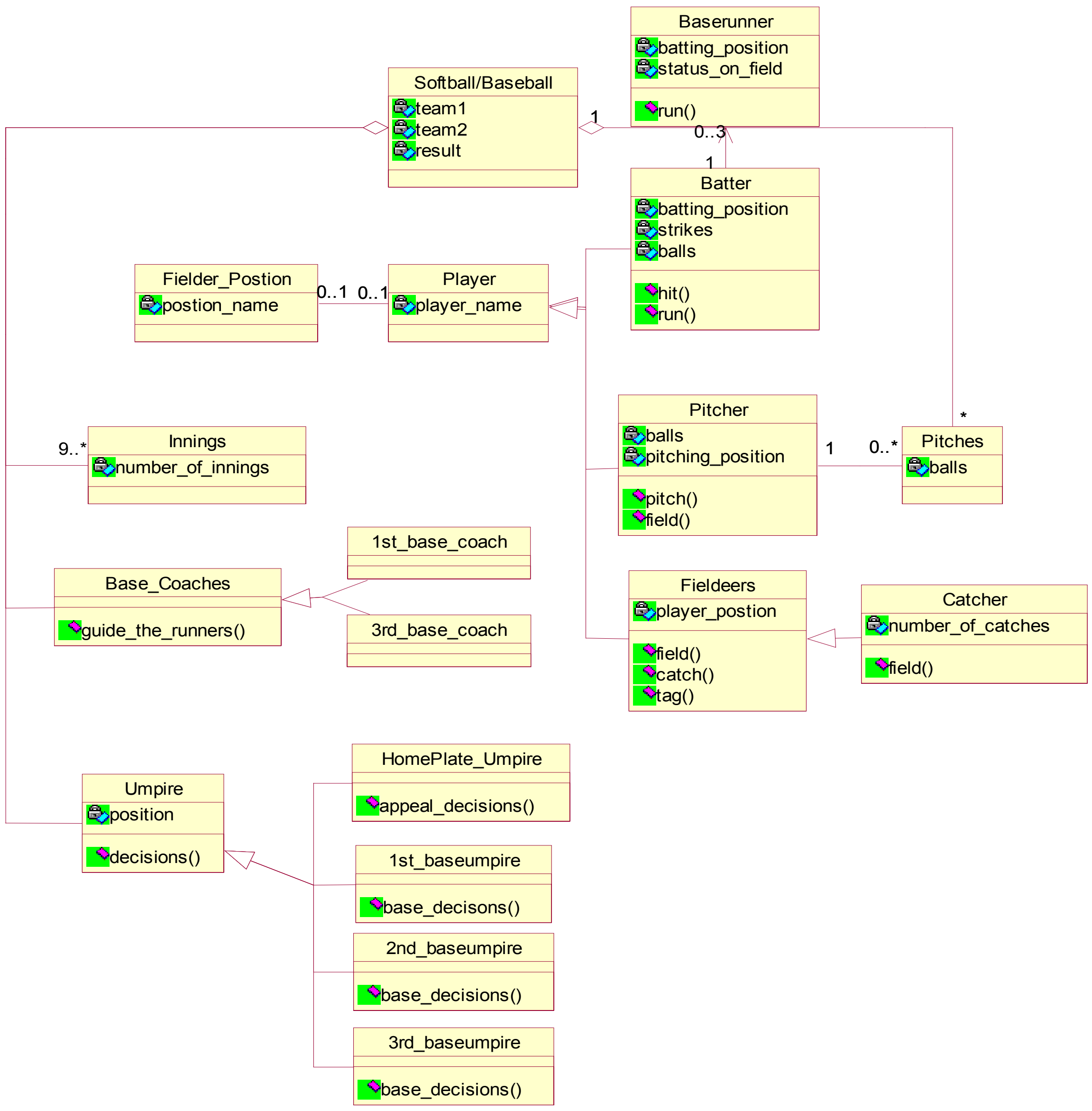

Figure 8: Class diagram for Baseball and Softball games

REFERENCES

[1]. Model Driven Architecture (MDA) Document number ormsc/2001-07-01.

[2]. Developing in OMG's Model Driven Architecture Jon Siegel and the OMG Staff Strategy Group.

[3]. MDA Explained Chapter 1: The MDA Development Process.

[4]. http://uml.omg.org/ UML 1.5 documentation's UML Summary.

[5].http://www.microgold.com/stage/UML_FAQ.html

[6]. http://www.sei.cmu.edu/domain-engineeering

[7]. "Domain analysis concepts and research direction" by Guillermo Arango

[8]. http://www.domain-specific.com/Processes.html