Model-driven Development in C

A C Ojha*

Model Driven Development (MDD) is rapidly gaining popularity in software development. The Unified Modeling Language (UML) is accepted as the industry standard modeling language to support it. Companies are realizing the benefits from this approach by reducing development time while producing higher quality software meeting customer expectations through removing defects in the early stages of software development. However, the application of UML is well known in object-oriented programming languages and very little or not known in function-oriented programs. This article presents a mapping of UML to C programming constructs and explains how to apply UML for modeling software applications developed using C programming language to get benefits of MDD.

Introduction

The increasing complexity of software systems is posing many challenges for software engineers. Document-driven approach to software development that has been a predominant approach in the past falls short to address these challenges. Although document-driven approach has been a bit successful, it fails to unambiguously capture the system requirements which leads to a system developed not meeting the customer expectations most of the time.

The problem of requirements specification has been attempted to solve with formal approach to software development. Formal approach specifies the system precisely and unambiguously. The system specifications can be verified and validated automatically for correctness. It can produce executable code with tool support. However, formal languages such as Z, which mostly take their basis from mathematics, are inherently complicated to interpret and use by software engineers and customers of the software system. Thus, the approach does not find a better place in the software industry for widespread acceptance of it.

The emerging Model Driven Development (MDD) is an OMG (Object Management Group) initiative that provides an approach to system development based on model definition and transformations. Models are used throughout the software

* Faculty Member, The ICFAI School of Information Technology (ISIT), Hyderabad. Email: acojha2002@yahoo.co.in

development life cycle for analysis, design, construction, deployment and maintenance. A model is a description of a system written in a powerful and expressive language. The language has a well-defined syntax and semantics, which is suitable for automated interpretation and manipulation. The OMG's Unified Modeling Language (UML) has become the most widely used standard for specifying and documenting a software system through several models. It is a visual and general purpose modeling language that can be applied to all the application domains and implementation platforms. Its extensive nature allows a software engineer to model software systems with all the major development methodologies. However, UML seems to be very popular where object-oriented methodology is used for software development.

The C is a general purpose programming language well known for powerful function-oriented programming. Software developers, who are content with C, use informal structured analysis and design methodology to develop systems. With rising popularity of modern object-oriented programming languages like Java and C#, object-oriented analysis and design methodology have become the mainstream approach for software development. Still a large chunk of software developments use C programming language with traditional document driven structured analysis and design approach. In particular, the embedded software developers who are more accustomed to C programming language, experience tough challenges from today's increasingly complex embedded systems. As time-to-market issues archive greater focus, enhancement of developers productivity and quality of system are major concerns, software developers sought alternative approach to find relief. One elegant approach is to move to UML and MDD to accrue many visible benefits.

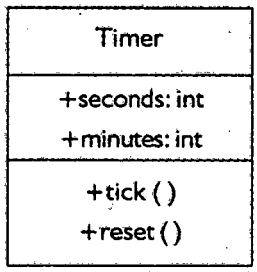
Models are used throughout the software development life cycle for analysis, design, construction, deployment and maintenance

The article is structured as follows. Section 2 describes the mapping of UML to C programming constructs. Section 3 discusses the benefits of MDD and then the article concludes with a positive note.

Mapping UML to C

Although UML is extensive and can be applied to any methodology, it is more biased towards object oriented design techniques. The basic concept of the object is easy to understand and adopt in C. While the key concepts of Object Oriented (OO) methodology like abstraction, encapsulation, data hiding, and code reuse can be effectively applied to C, the concept of inheritance seems to be complicated. Thus, a pragmatic subset of OO approach should be used to preserve its essence while implementing UML in a procedural language like C.

Figure 1: Class Diagram



Let's take the example of a Timer object that could have the responsibility to keep track of time. It could have attributes such as minutes and seconds and perhaps operations such as reset and tick. The reset operation could initialize the attributes to zero and the tick operation could increment the time by one second. In UML this could be shown in a class diagram having a single object called Timer that has attributes minutes and seconds and the type integer as well as public operations tick() and reset().

While the timer object is realized as a type class in an object oriented language, in C a structure type can be exploited for it. Thus, in C, the code should look like the following:

```
 type struct
{
 int seconds;
 int minutes;
} Timer;
void tick(Timer *this);
void reset(Timer *this);
```

Like normal C code, the attributes are contained inside the structure and the operations that act on these attributes kept outside. However, to distinguish which object these operations are to act upon, a pointer argument of type Timer is passed to these operations so that the attributes can be manipulated without any confusion.

Thus, the operations, for example, reset() and tick() can be defined as follows:

```
void reset(Timer *this)
{
 this->seconds=0;
 this->minutes=0;
}
void tick(Timer *this)
{
 this->seconds=this->seconds+1;
  if(this->seconds>=60)
{
  this->minutes=this->minutes+1;
  this->seconds=0;
  }
}
```

Initialization and clean up of an object is done with constructor and destructor functions which can be handled in C as well. A function init() to initialize the object upon creation and a function cleanup() to clean up the object upon destruction can be defined.

```
void init(Timer *this);
void cleanup(Timer *this);
```

An instance for an object can be created and destroyed statically or dynamically. However, the init() and cleanup() functions are not invoked automatically during static creation and destruction of the object. The static instance is very straightforward and is done as follows:

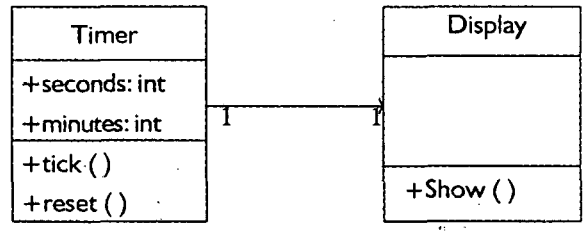
```
Timer time;
```

The instance is destroyed automatically when created statically. The dynamic creation and destruction of instance can be handled by creating create() and destroy() functions that allocate and release memory manually.

```
Timer * create()
{
 Time *this;
                              this
=(Timer*)malloc(sizeof(Timer));
 init(this);
 return this;
}

void destroy(Timer *this)
{
  cleanup(this);
  free(this);
}
```

Figure 2: Association Relation

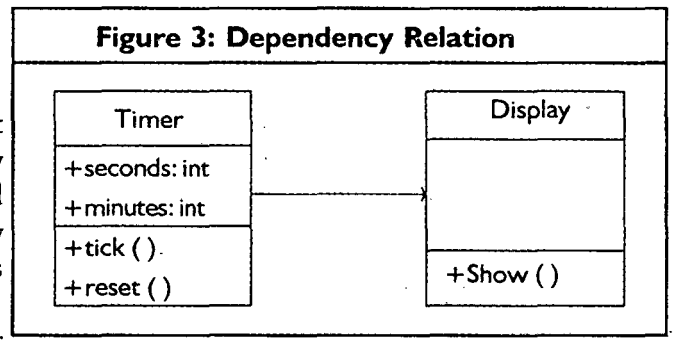


In UML the operations can be public or private. This can be achieved by using two different files for an object, a specification (.h) file and an implementation (.c) file. The private functions that are only visible to the object are declared and defined in the implementation file, but not available in the specification file. The public functions are declared in the specification file but defined in the implementation file. The specification file is included whenever the use of the object is required in a file. Thus, it preserves the fundamental concepts of abstraction, encapsulation, hiding and reuse.

In UML, the objects exploit some sort of relationship in order to interact with each other and invoke each other's functions. For example, the Timer object interacts with a Display object to show the time. If the Timer object has to invoke the show() function of the Display object, then an association relation from the Timer to the Display object can be established. Also a multiplicity can indicate how many instances of the Display object the Timer requires. The UML diagram will be as Figure 2.

The resulted C code for the Timer object is:

```
type struct
{
int seconds;
int minutes;
Display *display;
} Timer;
```



If the multiplicity is a constant greater than one, then an array of pointers of type Display could be used. If the multiplicity is many and varies with time then perhaps a linked list is required.

The function call for show(), inside the function tick() of Timer object could be as follows:

```
tick(Timer *this)
{
show(this->display, this->seconds, this->minutes);
}
```

In this case, the Timer object has an attribute that is an instance of the Display object. It is a structural relationship where one object contains an instance of another object. In certain situations a non-structural relationship holds between the two objects. It is called a dependency relation. In a dependency relation, one object uses another object by accepting it as an argument in the signature of its operation. The argument can be passed by value or by reference. The UML diagram for the Timer and the Display objects holding a dependency relation looks like as shown in the Figure 3.

The C code for the tick() function of the Timer object would be:

```
tick(Timer *this, Display *device)
{
show(device, this->seconds, this->minutes);
}
```

A summary of these mappings is presented.

- The basic concept of object is mapped to a structure in C that uses two files, a specification (.h) file and an implementation (.c) file. The specification file holds the structure definition having the attributes of the object as part of the structure.

The functions that operate on these attributes are declared outside the structure and defined in the implementation file. Other objects use the specification file to gain access to that object through its public interfaces.

- All the private and public instance attributes are declared in the structure. Mutator and accessor functions to change these attributes can only be used and made private or public as required. This discipline has to be maintained to implement data hiding.
- Class attributes, if any, are declared in the implementation file. Private class attributes do not appear in the specification file. They are marked as static in the implementation file. Public class attributes are declared bare in the implementation file and marked as extern in the specification file.
- Instance operations of the object have a standard first parameter called 'this', a pointer to the object data. Class operations are just normal functions in C. Private operations do not appear in the specification file. They are defined in the implementation file only and marked as static. Public operations are declared in the specification file and defined in the implementation file.

Model simulation and code generation enable faster design iterations that produce desired functionality, performance and scalability

(A case study using these concepts is provided separately in Developers' Zone in this issue.)

Benefits of MDD

Model-driven development is a significant shift in software development that promises many improvements of the software development process to deliver business benefits.

- Improved Productivity: MDD supported by tool can generate code automatically through model transformation. In fact a tool can generate code between 65-90% of the entire system from the models and remaining 10-35% is the code that the developers write for function bodies. This is a significant improvement of developers' productivity during initial system development, refinement and system maintenance. This enables the developers shifting focus from coding to modeling, thus paying more attention to solving the business problem at hand.
- Better Quality: MDD produces quality code by assuring conformance to requirement specifications, removing construction defects through model validation, refinement, and simulation, maintaining uniformity and consistency through the use of coding standard. Model simulation and code generation enable faster design iterations that produce desired functionality, performance and scalability.
- Reduced Cost and Time: Automatic code generation removes repetitive and mechanical parts of the coding process and reduces the manpower requirements. It enables low development cost and reduced time-to-market the software. Development process becomes more predictable thereby lowering the project risks.

- Increased Business Agility: MDD helps respond to the changing business requirements quickly by modifying the models or code easily to fit in changes. This is due to the fact that the mapping between the model and code is kept very straightforward and roundtrip-engineering tool makes the correct synchronization. It helps in better maintenance. A business-focused approach to software development produces software that meets business agility.
- Documentation: Since model describes the system at different levels of abstraction, they are used for better documentation and communication. Thus documentation process is also automated.
- Solving Complexity through OO Concepts: Object-oriented paradigm itself has several benefits over function-oriented approach. Abstraction, encapsulation, inheritance and reuse are the key OO concepts. They reduce complexity of developing large systems. They help provide better structured, more extensible and maintainable systems. A 'C' developer can leverage these benefits.

Conclusion

Models support software development at a higher level of abstraction bringing greater communication, maintainability and flexibility. MDD can solve many pains of C developers through requirement visualization, model simulation and quality code generation from the validated models. It requires a shift in culture from document driven code-centric development to model-centric development. UML models and OO concepts are basic building blocks for this approach. However, the technical complexity of UML modeling and a different approach to software development in OO paradigm throw some concerns for C developers. They find difficulty in transition from a C development environment to an OO development environment for adoption of MDD. Still then, the benefits of MDD are significant and the C developers must cope with the transition to experience it.

Reference # 35J-2005-12-01-01

References

1. Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addition-Wesley, 2000.
2. Byron S Gottfried, *Theory and Problems of Programming with* C, Schaum's Outline Series, McGraw Hill Publishing Company, 1990.
3. Richard Soley, "Model Driven Architecture", OMG white Paper, November, 2000, at http://www.omg.org.
4. Marin Bakal, "UML for C Programmers", C/C++ *Users Journal*, May 2005.
5. Mark Richardson, "Object Based Development Using the UML and C", *Dedicated System Magazine*, 2001.
6. Website at http://www.ilogix.com

Learning for Leadership

ICFAI UNIVERSITY PRESS Book Club

ICFAI University Press publishes over 300 new books, every year that are focused to impart relevant and contemporary knowledge and cutting edge skills to corporate executives, consultants, academicians and students. You can learn to win and lead !

JOIN NOW
Free Books Worth Rs.3,000
MEMBERSHIP

for online registration
Visit us at :
www.icfaipress.org/bookclub

Join Today & Enjoy these benefits for 5 years

Highlights

- Four Categories of Membership: Standard, Silver, Gold and Platinum.
- Online Membership Registration Facility.
- Membership valid for 5 years.
- Big savings now-Join today and select up to 10 books Free.
- Growing catalogue of books.
- Choice of the latest Books on Management, Finance, IT & allied areas.
- Fortnightly electronic newsletter.
- Books at special discounts exclusively for members.
- Free books through Booklovers scheme.
- Savings on Subscription of Magazines and Journals.
- Easy payment options through Online/Cheque/DD/Credit Card(VISA/MasterCard).
- Customer Service through All India Network.
- Free Delivery of Books at your doorstep.

Membership Categories

Category	Membership Fee (Rs.)	Free Books	
		Number	Value(Rs.)
Standard	500	3	900
Silver	750	5	1500
Gold	1000	7	2100
Platinum	1250	10	3000

On subsequent purchase of books, members receive discounts : Standard (33%), Silver (40%), Gold (45%) and Platinum (50%).

To seek membership in ICFAI University Press Book Club, please contact:
The Manager, ICFAI University Press Book Club, 52,
Nagarjuna Hills, Punjagutta, Hyderabad 500082.
Ph: 040-2343 5368/6970/72-74. Fax: 040-2335 2521/4302.
E-mail: bookclub@icfaipress.org