

## FORMAL METHODS FOR QUALITY OF SERVICE ANALYSIS IN COMPONENT-BASED DISTRIBUTED COMPUTING

**Chunmin Yang, Barrett R. Bryant, Carol C. Burt**

Department of Computer and Information Sciences, The University of Alabama at Birmingham,  
Birmingham, AL, U.S.A.

**Rajeev R. Raje, Andrew M. Olson**

Department of Computer and Information Science, Indiana University Purdue University  
Indianapolis, Indianapolis, IN, U.S.A.

**Mikhail Auguston**

Department of Computer Science, Naval Postgraduate School, Monterey, CA, U.S.A.

*Component-Based Software Architecture is a promising solution for realizing distributed computing systems. To develop high quality software for such systems, an analysis of non-functional aspects of the software properties (also called Quality of Service or QoS) is very important. The UniFrame research project defines a Unified Meta-Component Model Framework (UniFrame) that includes a notion of QoS contracts. A classification of Quality of Service parameters, both static and dynamic, relevant to component-based distributed computing is developed and represented formally using Two-Level Grammar (TLG), an object-oriented formal specification language. TLG may be transformed into both a UML model, augmented with OCL constraints, and executable code in the Java programming language. This may be regarded as standardized code for implementation of the distributed application with dynamic measurement of the Quality of Service aspects incorporated. The approach is consistent with OMG's Model Driven Architecture (MDA) in that QoS properties may be specified at the Platform Independent Model (PIM) level and then carried down to the Platform Specific Model (PSM) level in implementation.*

**Keywords:** *Formal Specification, Quality of Service (QoS) Component-Based Software Architecture, UniFrame, Two-Level Grammar (TLG), Object Constraint Language (OCL), Model Driven Architecture (MDA).*

### 1. Introduction

Component-Based Software Architecture (CBSA), a viable and economical alternative to traditional software design, is also a promising solution for realizing software for distributed computing systems. Components, by definition, are independent of the language implementation, tools and the execution environment. In practice, the systems to be modeled and implemented, and the environment that these systems are situated in, change frequently. UniFrame<sup>1</sup> is a unified framework (Raje et al., 2001) that allows a seamless integration of heterogeneous and distributed software components. Each component created using the UniFrame approach has a Unified Meta-component Model (UMM) specification (Raje, 2000). The core parts of the UMM are: components, service and service guarantees, and

---

<sup>1</sup> <http://www.cs.iupui.edu/uniFrame>

infrastructure. Non-functional properties, also called Quality of Service (QoS)<sup>2</sup> are important aspects of UMM specification.

In this paper, CBSA and formal specifications are used to specify non-functional properties, and to convert the natural language requirements of the non-functional properties into application programs. In this way, the non-functional aspects of the software systems are considered and integrated into the system, just like the functional aspects. Since the system is modeled in a formal way, it is easier to monitor and maintain. Since the system will be developed using Model Driven Architecture (MDA)<sup>3</sup> and the Unified Modeling Language (UML)<sup>4</sup>, the Object Constraint Language (OCL)<sup>5</sup> is integrated into UniFrame to specify the QoS requirements of the distributed computing applications. As an extension of UML, the OCL is powerful and efficient in specifying the constraints on the components that cannot be easily specified by UML diagrams. The OCL is a formal language, and is compatible with the CBSA. Thus, with the OCL specification, the QoS requirements can be specified in a formal way and automatically weaved into the generated code.

The rest of the paper is organized as follows. The next section describes the motivation for our research related to the Quality of Service analysis. The formal language we use is briefly introduced in section 3. Section 4 describes the technical basis for our specification and representation of the QoS approach. The overall structure of the specification and automatic conversion is described with a simple example in section 5. This example focuses on the integration of OCL and MDA in the QoS analysis and associated automation. This paper ends with conclusions and future work.

## 2. Quality of Service

In Distributed Component Systems (DCS), the Quality of Service (QoS) aspects are as important as the functional aspects. Quality of Service is a concept originated in the networking area and has been extended to software development. However, QoS concepts are complex, abstract, not quantifiable, and difficult to specify and model (Yang et al., 2002). The effect of QoS properties on the system does not necessarily remain the same all the time (Rosa et al., 2002). It is especially difficult to model and formulate these properties during the early stages of the software development. In addition, QoS specifications are rarely supported by computer languages, methodologies, or tools (Pal et al., 2000). Hence, it is not easy to decide if the system meets the QoS requirements until the latter half of the software development phase. It is even harder to validate the non-functional properties of software, as there are not many well accepted models for the quantification of QoS parameters. Because of all these reasons, it is more difficult to specify the QoS parameters than to specify the functional aspects of the software requirements.

However, to develop high quality software, QoS properties have to be taken into consideration. There have been several research projects with this goal, e.g., Aster (ASTER, 2000), Qedo<sup>6</sup>, QuO<sup>7</sup> (Pal, 2000), to name a few, but not many attempts have been made to incorporate QoS into component-based software systems.

Our goal is to describe the QoS parameters with a formal language to standardize the software development of systems in which QoS is integrated. Natural languages are too informal and ambiguous for precisely describing QoS requirements, but on the other hand, programming languages are not appropriate either because of too much detail, and the platform dependency of the execution environment of high level programming languages. Formal specifications can overcome both of these

---

<sup>2</sup> The terms “Non-Functional Properties” and “Quality of Service (QoS)” are used interchangeably in this paper.

<sup>3</sup> <http://www.omg.org/mda/>

<sup>4</sup> <http://www.omg.org/uml/>

<sup>5</sup> <http://www-3.ibm.com/software/ad/library/standards/ocl.html>

<sup>6</sup> <http://qedo.berlios.de>

<sup>7</sup> <http://www.dist-systems.bbn.com/tech/QuO>

problems (Yang et al., 2002) and can also be elegantly integrated into component-based software development techniques.

In order to specify and analyze the QoS properties, they are divided into three aspects: non-functional attributes, non-functional actions and non-functional properties (Yang et al., 2002). Non-functional attributes are the features to be specified, a significant characteristic of which are their decomposability in the sense that a non-functional attribute can be decomposed into more detailed attributes. Non-functional actions are the inputs that have effects on the non-functional attributes. Non-functional properties are the constraints of the non-functional actions over the non-functional attributes. A simple example of these aspects could be: the response time of a distributed system is a non-functional attribute, and the clients connecting into the system and disconnecting from the system are possible non-functional actions. Non-functional properties define how the connecting or disconnecting operation may affect the non-functional attributes and what kind of response time is expected in this system.

Four steps are taken to assure the QoS of a Distributed Computing System: first, create a catalog for the QoS parameters; then provide a formal specification of them; third, construct a mechanism to guarantee the specified values for these parameters, both at the individual component level and at the entire system level, both statically (by prediction) and dynamically (by empirical evaluation and monitoring); last, perform testing to make sure the constructed system meets the original requirement specification, especially with respect to QoS.

A catalog of Quality of Service parameters is described in (Raje et al., 2002). It includes many parameters such as security, throughput, capacity. The format of this catalog is based on the format of the design patterns catalog (Gamma et al., 1995), for both static and dynamic properties of the QoS parameters.

A number of architectures have been proposed for QoS guarantees for distributed systems, for example, the Quality Objects (QuO) framework. This work mainly emphasizes specification, measurement, control and adaptation to changes in Quality of Service.

In UniFrame, we formally specify the quality of components and component complexes (results of compositions of components). They are specified and verified in the context of combining heterogeneous components. This provides a QoS management approach to the interactions between interacting components in distributed systems by supporting frameworks for multiple QoS categories (Raje et al., 2002). The following features of the UniFrame approach, for QoS, distinguish it from other related efforts:

1. The creation of a QoS Catalog for software components containing detailed descriptions about QoS attributes of software components including the metrics, evaluation methodologies and the interrelationships with other attributes.
2. An integration of QoS at the individual component and distributed system levels.
3. Formal specifications, based on Two-Level Grammar.
4. The validation and assurance of QoS, based on the concept of event grammars (Auguston, 2000).
5. An investigation of the effects of component composition on QoS; involving the estimation of the QoS of an ensemble of software components given the QoS of individual components.
6. The automatic translation from natural language to formal specification languages, and then to UML class diagrams and application programs.
7. An integration of OCL to specify the QoS properties and to convert them to high level programming languages.
8. The conformance to the OMG MDA standard by specifying QoS in Platform Independent Models and implementing it in Platform Specific Models.

### 3. Two-Level Grammar

Since it is a formal specification language at a level of abstraction between natural and programming languages, in UniFrame, we use Two-Level Grammar (TLG) (Bryant and Lee, 2002) to specify QoS parameters, and convert them into a UML specification with integrated OCL constraints and conforming to MDA. TLG is a formal specification language, originally developed as a specification language for programming language syntax and semantics (van Wijngaarden, 1974), and later used as an executable specification language and as the basis for conversion from requirements expressed in natural language into formal specifications (Bryant and Lee, 2002). It is a formal notation based upon natural language and the functional, logic and object-oriented programming paradigms. The combination of natural language and formalization is unique to TLG and also fits the Unified Meta-component Model (UMM) (Raje, 2000) for component description used in UniFrame well. The specification in TLG is easy to read and may be automatically generated from natural language requirements specifications (Lee and Bryant, 2002a).

The name “two-level” in TLG comes from the fact that TLG consists of two Context Free Languages defining the set of type domains and the set of function definitions operating on those domains, respectively. These grammars may be defined in the context of a class in which case type domains define instance variables of the class and function definitions define methods of the class, and they interact with each other to achieve the power of a Turing Machine (Yang, et al., 2002).

The syntax of TLG class definitions is:

```
class Identifier-1 [extends Identifier-2, ... Identifier-n].  
    instance variable and function declarations  
end class [Identifier-1].
```

The function signature is defined as follows:

Function signature : function-call-1, ..., function-call-n.

TLG is suitable for representing QoS properties because with its class hierarchy that corresponds to the way we describe QoS properties (Yang, et al., 2002), we can take advantage of CBSA and component reuse. Especially since it supports multiple inheritance, it may be used to represent the decomposability of the QoS properties. The instance variables and functions can be used to represent the QoS attributes and associated actions. TLG has a high level of abstraction and its representation is flexible – not all the members have to be quantifiable, and this suits the feature of QoS properties since most of the effects of the QoS are either existent or not existent, instead of being a quantifiable concept.

### 4. OCL, MDA and Quality of Service

The specification of the OCL is a part of the UML specification, and it is not intended to replace existing formal languages, but to supplement the need to describe the additional constraints about the objects that cannot be easily represented in graphical diagrams, like the interactions between the components and the constraints between the components’ communication. In object-oriented modeling, a graphical model, such as a class diagram, is not enough for a precise specification of the components and their interaction. OCL is designed to solve this problem. It facilitates the specification of model properties in a formal yet comprehensive way. By combining the power of the straightforward, graphical UML modeling and the textual, accurate OCL constraints, information can be specified in a formal way.

OCL has the characteristics of an expression language, a modeling language and a formal language. An OCL expression is guaranteed to be without side effects since it is an expression language, and

thus, cannot change anything in the model. However, an OCL expression can be used to specify the state changes of the system. OCL is not a programming language, but a modeling language. So, it is impossible to write program logic or flow-control in OCL (Neema et al., 2002). All implementation issues are out of the scope of OCL. OCL is also a formal language where constructs have a formally defined meaning; in other words, it is unambiguous. Furthermore, OCL is strongly typed.

The main idea behind OCL is “Design By Contract” (DBC) (Frankel, 2003). By applying this, the responsibility of the parties can be formally described. An OCL constraint consists of the precondition, the postcondition and the invariant. The contract is a way of establishing who does what by stating, first, what must be true for the caller (say, client, for example) to request a service from the callee (server, for example), and, what must be true when the callee finishes providing the service. The former is the precondition and the latter is the postcondition. The invariant must be true when a routine is called and when it terminates, but not necessarily when it is executing. By the principle of “Design By Contract”, and specifying these three constraints, the services provided by the server are exposed, but not the details of the implementation of the services.

On the other hand, the callee will know when exactly a service can be provided (available), and the caller will know when exactly it can request the service. In case of exceptions, it is easy to find out who caused the exception: if the precondition is false, the caller broke the contract; if the postcondition is false, the callee broke the contract; if the invariant is false, the callee class broke the contract.

Since OCL is a textual extension of the graphical UML modeling language and OCL specification is always unambiguous, so it adds better documentation to the visual models. It can be used during the modeling and specification. Since OCL is an expression language, it can be checked without an executable system. All these features turn out to be useful in representing QoS properties, which can be represented by the combination of precondition, postcondition and invariant in OCL. The QoS attributes are represented by the member variables of the class, and the QoS actions are represented by the methods. They are checked at run time, before and after the calls so that the change of the QoS parameters of the system is monitored in a timely manner.

The precondition has to be satisfied before the method can be called, and the postcondition has to be satisfied by the time the method returns. It is easy to find out which step causes exceptions if any. The methods are called in a loop-like fashion, so, whenever a change of the QoS parameter is observed (by some technique), appropriate methods are called to make the necessary change notifications. The QoS specification is integrated in the overall system design in this fashion. In this way, the satisfaction of the QoS requirements is guaranteed and the change of the QoS properties is under observation and control, as well.

Although QoS properties and associated metrics have been widely used in networking, there is no standard vocabulary for discussing the QoS as it relates to the distributed computing and component-based solutions (Burt et al., 2002), especially when the QoS properties are applied on various platforms and when the different aspects of the QoS interact with each other. A standard vocabulary is the first step toward advancing Model Driven Architecture that includes QoS parameterization and/or QoS contracts. This is one of the goals of the UniFrame project.

MDA provides an open, vendor-neutral environment for the integration of different distributed application software. MDA aims to separate the business or application logic from the underlying platform technology. Its standards are made up of the Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI), and Common Warehouse Meta-model (CWM) (Frankel, 2003). Platform-independent applications built using MDA and the associated standards can be realized on a range of platforms.

MDA has standards that enable the use of generative techniques for the construction of interoperability bridges between different platform technologies. In a distributed environment, it is normal to see different components of the system running on dispersed and different platforms, and using various techniques. By applying the MDA architecture, the detailed differences are hidden from

the application layer. This is especially useful for the modeling, analysis and control of QoS of the systems.

The MDA design initiative assists during the interaction between the different platforms and different middleware. Middleware environments started out providing the interoperability using the architectures that are standard, proprietary, or somewhere in the middle. Progressively, more and more services and more powerful middleware have been added to the overall architecture, thus, it is more difficult to ensure the interoperability of this middleware. To efficiently solve this problem, MDA is designed by applying the component and modeling technology and putting the whole picture together.

There are several core models in MDA: one represents enterprise computing with its component structure and transactional interaction; another represents real-time computing (for which QoS is an important aspect) with its special needs for resource control; and some others that represent specialized environments. Each of these models will be independent of any middleware platform.

MDA defines two models: Platform Independent Model (PIM) and Platform Specific Model (PSM) and the conversion between the PIM and the PSM. A PIM describes the business processes and entities in terms of components, and does not specify the implementation of the software system as such. A PSM, however, describes how to build the components by applying mapping profiles that target different software technologies. It works together with the domain business information model and some other details. Hence, the PIM and the PSM separate the design model from the implementation model by providing multiple layers, each of which focuses on a different level of abstraction and platform and domain information.

The first step when constructing an MDA-based application will be to create a platform independent application model expressed via UML in terms of the appropriate core model. Adding new middleware platforms to the interoperability environment is straightforward: after identifying the way a new platform represents and implements common middleware concepts and functions, this information is incorporated into the MDA as a mapping.

## **5. OCL/MDA Integration for QoS Analysis**

UML has been popularly used in object-oriented design and is useful for modeling systems, their behavior and interaction. However, UML currently does not support the modeling of QoS properties of objects or components and there has been no special attention to modeling quality requirements or to expressing in UML QoS aspects of software architectures (UML, 2002).

We present a formal semantics for the object constraint language that is part of the UML. In the context of information systems modeling, UML class diagrams can be utilized for describing the overall structure, whereas additional integrity constraints and queries are specified with OCL expressions.

The Generic Modeling Environment (GME) (GME, 2001) is a configurable, domain-specific, model-integrated tool for creating and evolving domain specific, multi-aspect models of systems developed by the Institute for Software Integrated Systems (ISIS) at Vanderbilt University<sup>8</sup>. OCL is embedded in GME to specify the constraints of the interactions between different components of the system to be modeled (Gray, 2001).

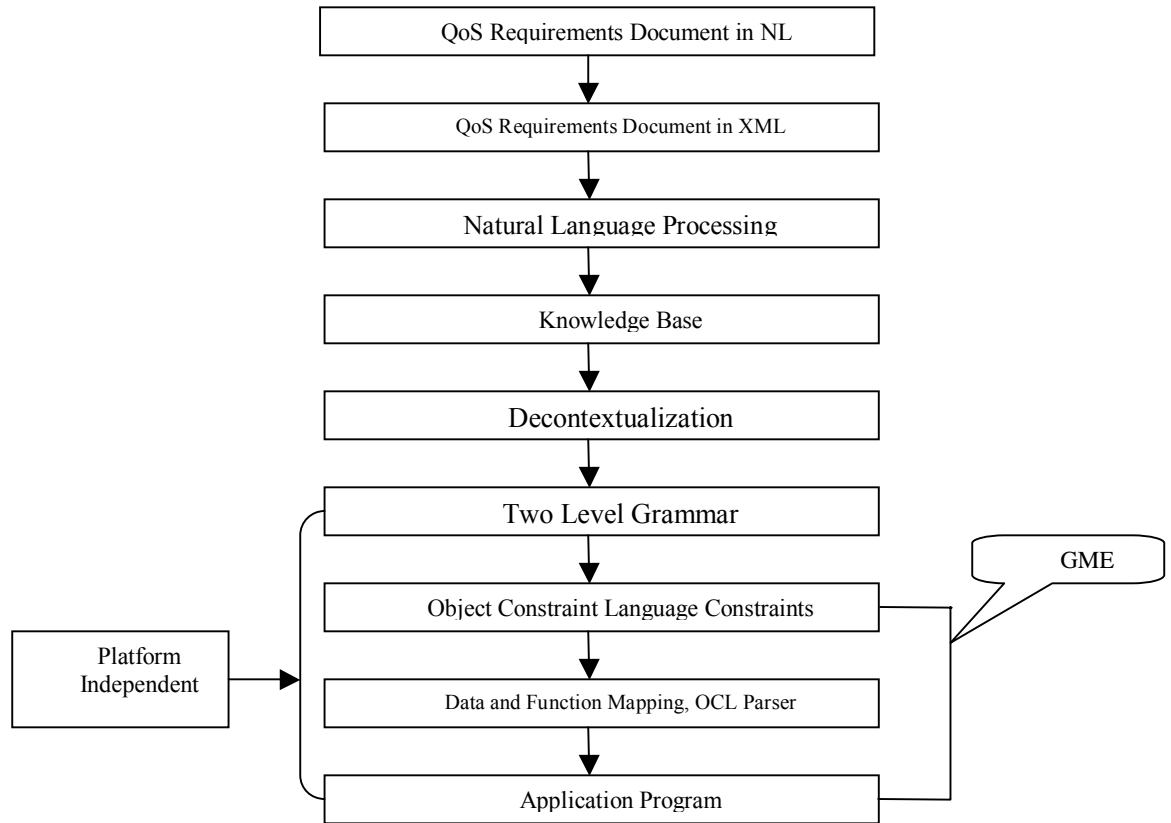
GME uses the technique of Model Integrated Computing (MIC) (Nordstrom, et al., 1999). MIC is a methodology for generating application programs automatically from multi-aspect models. GME automatically generates the meta-level specification, and does not depend on the specific domain the application is in. Since GME supports the specification of OCL constraints, we can use this tool to specify our QoS specification in OCL and integrate with the other aspects of the software requirements, and generate the application level programs, as either UML models, or object-oriented programs. With the idea of Platform Independent Modeling in MDA, if we specify the QoS of a system in a way that

---

<sup>8</sup> <http://www.isis.vanderbilt.edu>

conforms to the MDA standard, this specification will be able to be applied to any platform without worrying about the difference of details in the domain or platform environment to which this specification is applied.

The conversion process from a natural language requirements document into executable code is shown in Figure 1.



**Figure 1. System Structure**

To illustrate this process, we use a simple ATM example to demonstrate the process of converting the QoS specification from a natural language into a modeling language (Lee and Bryant, 2002a). First, we start with a natural language description of the QoS requirements. A brief description of the QoS requirements of the ATM (Yang et al., 2002) is reprinted here:

ATM's security property is as follows. The length of the encryption byte should be bigger than 3 and the allowed attempts has to be smaller than the maximum allowed attempts. If the encryption byte length is 6 and the maximum allowed attempts is less than 5 then the system is 80% secure. If the account type is a savings account or the maximum allowed connections of the bank is less than 50 or the delay level is less than 50 then the maximum allowed attempts is limited to 4.

If the user timeout is between 10000 and 120000 milliseconds we have a good delay level. If the response time is longer than 30000 milliseconds, the delay level drops down to 40%.

As can be seen, we are mainly concerned with “security” issues in this example. Since TLG is a natural language-like formal specification language, it is easy to read and it is easier to convert the natural language specification to TLG than to convert to other formal specification languages. To convert from a natural language description to a TLG specification, the QoS properties description is first represented in XML to specify which role each sentence plays.

A sample XML representation of the ATM example is shown as follows.

```

<document>
  <c title="ATM">
    <c title="Security">
      <p meta="satisfaction check">
        <s>The length of the encryption byte should be bigger than 3 and the allowed attempts has to be
          smaller than the maximum allowed attempts
        </s>
      </p>
      <p meta="level update">
        <s>If the encryption byte length is 6 and the allowed attempts is less than 5 then the system is 80%
          secure
        </s>
      </p>
      <p meta="attribute update">
        <s>If the account type is a savings account or the maximum allowed connections of the bank is
          less than 50 or the delay level is less than 50 then the maximum allowed attempts is limited to
          4
        </s>
      </p>
    </c>
    <c title="Delay">
      <p meta="satisfaction check">
        <s>If the timeout is between 10000 and 120000 milliseconds we have a good delay level
        </s>
      </p>
      <p meta="level update">
        <s>If the response time is longer than 30000 milliseconds the delay level drops down to 40%
        </s>
      </p>
    </c>
  </c>
</document>

```

Titles such as Security and Delay indicates the property types whereas the meta information such as satisfaction check, level update, and attribute update indicates the non-functional actions within the property.

Given this XML representation of QoS, each sentence of the specification is tokenized and then by using computational linguistics parsing techniques the system constructs its parsing tree (Lee and Bryant, 2002b). Based on the parsing tree and the meta information from the XML tags, a Knowledge Base is constructed. The Knowledge Base is an explicit and declarative representation that is used to represent, maintain, and manipulate knowledge about QoS of the system. This Knowledge Base is converted into TLG by identifying the classes, data types, and operations as shown below:

```

class Property.
  Level :: Integer.

```

end class.

```
class Bank_Capacity extends Property.  
    Maximum_Connections :: Integer.  
end class.
```

```
class ATM_Security extends Property.
```

```
    Maximum_Allowed_Attempts, Encryption_Byte_Length, Allowed_Attempts :: Integer.  
    Account_Type :: String.
```

```
check satisfaction:
```

```
    Encryption_Byte_Length > 3,  
    Allowed_Attempts < Maximum_Allowed_Attempts.
```

```
update level:
```

```
    Encryption_Byte_Length = 6;  
    Allowed_Attempts < 5,  
    Level := 80.
```

```
update attributes:
```

```
    Account_Type = "savings",  
    Maximum_Allowed_Attempts := 4;  
    Bank_Capacity.Maximum_Connections < 50,  
    Maximum_Allowed_Attempts := 4;  
    ATM_Delay.Level < 50,  
    Maximum_Allowed_Attempts := 4.
```

end class.

```
class ATM_Delay extends Property.
```

```
    Response_Time, User_Timeout :: Integer.
```

```
check satisfaction:
```

```
    User_Timeout > 10000, User_Timeout < 120000.
```

```
update level:
```

```
    Response_Time > 30000, Level := 40.
```

end class.

Each property is defined as a TLG class whereas the non-functional attributes are defined as TLG instance variables such as Level, Maximum\_Connections, Maximum\_Allowed\_Attempts, Encryption\_Byte\_Length, Allowed\_Attempts, Account\_Type, Response\_Time, and User\_Timeout. In our example, ATM has Security and Delay properties and Bank has Capacity property which is used in the update attribute operation of ATM\_Security. As the above TLG specification illustrates, all the property classes extend the class Property which has the instance variable Level. This variable is a representative value for the property, with which the decomposability of QoS is implemented. For Example ATM Security property has several attributes such as Encryption\_Byte\_Length and Allowed\_Attempts. The value of Level for ATM\_Security represents the overall security level after evaluating all the attributes.

As a formal language, OCL can be used to represent the constraints of the QoS properties. Our next step is to convert the TLG specification of QoS into an OCL specification. An OCL code snippet is listed here:

```
ATM_Security
  level: Integer
  encryption_byte_length: Integer
  allowed_attempts: Integer
  max_allowed_attempts: Integer
  account_type: String

  checkSatisfaction ()
  updateLevel ()
  updateAttributes ()

ATM_Security::checkSatisfaction ()
  Pre: encryption_byte_length > 3 and
      allowed_attempts < max_allowed_attempts

  Post: encryption_byte_length >=
      encryption_byte_length@pre and
      allowed_attempts <= allowed_attempts@pre

ATM_Security::updateLevel ()
  Pre: encryption_byte_length == 6 and
      allowed_attempts < 5
  Post: level = 80

ATM_Security::updateAttributes ()
  Pre: account_type == "savings"
  Post: max_allowed_attempts = 4
```

It can be seen that the Security property is checked and maintained by the preconditions and postconditions in OCL.

As indicated earlier, OCL is an expression language, so, OCL constraints cannot directly affect any models created using the target modeling language. The constraint expressions are merely formal comments on the semantics of the modeling language. Models created using the target modeling language can be verified using the OCL expressions, but the expressions cannot cause any changes in the models. This exactly suits our need to specify the QoS properties since it is commonly known that some of the QoS properties may change dynamically during the execution of the program, or because of some outside influence. OCL is perfect for representing these properties at the same time, as it does not change the properties. The syntax and semantics of TLG and OCL are similar to some extent which simplifies the conversion. The conversion from the TLG specification to the OCL constraints can be achieved by the mapping of the member variables of the TLG and of the OCL. Both TLG and OCL are strongly typed. The method conversion between these two specifications is achieved by the context analysis.

Using GME, we can parse the OCL constraints and generate the UML model or object-oriented program code, e.g., Java. Especially, one of the nice features of GME is the conversion between meta level and domain level as shown in Figure 1. After we convert the TLG specification of QoS into OCL, with the OCL parser, the application domain programs or models can be generated, regardless of the specific domain of this application. So the QoS properties are extracted from the domains and can all

be specified in a uniform way. The UML class diagram of the QoS specification of the ATM example is shown in Figure 2.

In this approach, the domain dependence can be masked by the GME tool; the platform dependent problem can be solved by integrating the MDA architecture. Since this approach will conform to the MDA standard, the specification of the QoS in this way is platform independent in the sense that the specification of the QoS parameters does not have any platform dependent information or constraints, so is applicable in any environment. In the distributed environment, when only the interfaces of the components are exposed, this QoS specification can be integrated into the overall system design. As far as platform independency is concerned, we are taking the MDA approach of converting from PIM to PSM.

The catalog of QoS parameters we create applies to all platforms, regardless of the programming language used to achieve it. In the design phase, the software system is designed in a platform independent manner in which the QoS properties are integrated as components. At the implementation phase, a platform dependent system instance is generated. In this way, the detail does not need to be considered, and there is no reason to let too much detail affect the design of the software. As long as we can specify the QoS properties in the design level, they are enforced at the implementation level, and they apply to various platforms, which is a very common need in distributed computing.

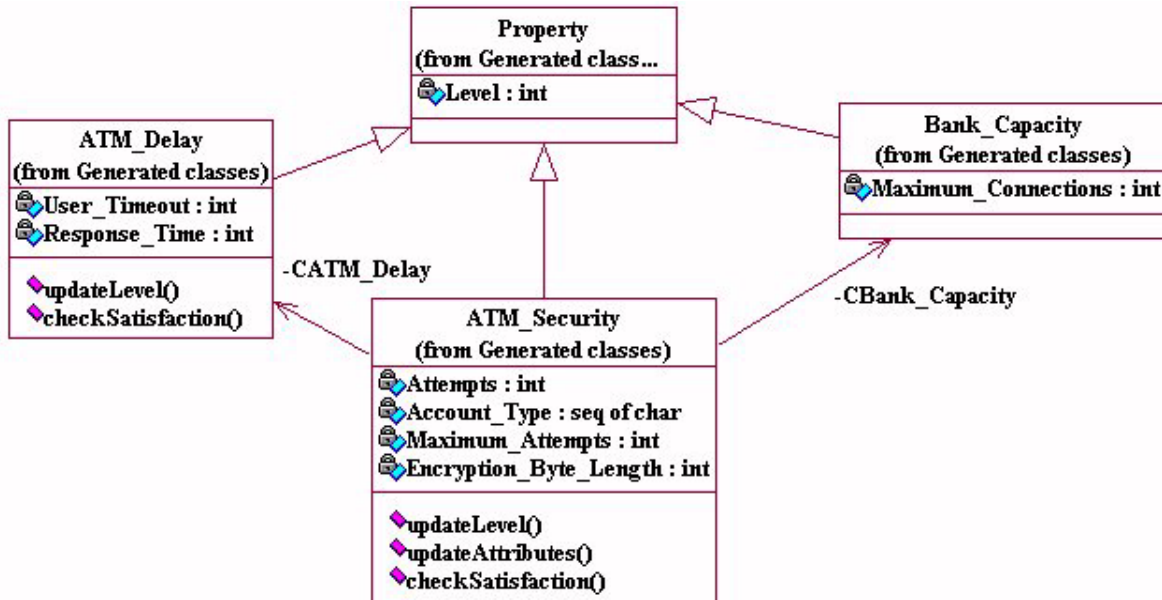


Figure 2. UML for QoS of ATM

## 6. Conclusion and Future Work

Quality of Service properties of the software requirements specification is an important part of software design consideration. In our research, the QoS requirements are described in natural language and later converted to UML or high level programming languages. By this way, developers working in different domains can specify their QoS requirements that will later be converted into a formal modeling language.

With the integration of OCL and MDA, more detailed requirements can be accurately expressed in the modeling stage of the software design, and the details of different platforms and operating systems

are hidden from the software designers and developers, which is especially beneficial in distributed applications. By applying the MDA tools, the Platform Specific Models of each domain or application can be generated from the Platform Independent Model, which integrates the QoS requirements and implementation in it.

Our future work is to automate the conversion of OCL, and implement the representation within MDA, and hopefully automate the standard documentation generation. At the same time, we plan to improve the compatibility of the conversion. The improvement of the usability of the system is another goal.

## 7. Acknowledgement

This material is based upon work supported by, or in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number DAAD19-00-1-0350 and by the U. S. Office of Naval Research under award number N00014-01-1-0746.

## 8. References

- ASTER, 2000. Software Architectures for Distributed Systems (ASTER). Technical report, (<http://www-rocq.inria.fr/slidor/work/aster.html>).
- Auguston, M., 2000, "Tools for Program Dynamic Analysis, Testing, and Debugging Based on Event Grammars". Proceedings of the 12<sup>th</sup> International Conference on Software Engineering and Knowledge Engineering, pp. 159-166.
- Bryant, B. R., and Lee, B-S, 2002, "Two-Level Grammar as an Object-Oriented Requirements Specification Language". Proceedings of the 35<sup>th</sup> Hawaii International Conference on System Sciences, [http://www.hicss.hawaii.edu/HICSS\\_35/HICSSpapers/PDFdocuments/STDSDL01.pdf](http://www.hicss.hawaii.edu/HICSS_35/HICSSpapers/PDFdocuments/STDSDL01.pdf).
- Burt, C. C., Bryant, B. R., Rajee, R. R., Olson, A., and Auguston, M., 2002, "Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models". Proceedings of EDOC 2002, the 6<sup>th</sup> IEEE International Enterprise Distributed Object Computing Conference, pp. 212-223.
- Frankel, D. S., 2003, "Model Driven Architecture: Applying MDA to Enterprise Computing", OMG Press.
- Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.
- GME, 2001, "Generic Modeling Environment: GME 2000 User's Manual", Version 2.0, Release 12-18-2001, Institute for Software Integrated Systems, Vanderbilt University.
- Gray, J., Bapty, T., Neema, S., Tuck, J., 2001, "Handling Crosscutting Constraints in Domain-Specific Modeling", Communications of the ACM, pp. 87-93.
- Lee, B.-S. and Bryant, B. R., 2002a, "Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language," Proceedings of SAC 2002, the 2002 ACM Symposium on Applied Computing, pp. 932-936.
- Lee, B-S and Bryant, B. R., 2002b, "Contextual Processing and DAML for Understanding Software Requirements Specifications," Proceedings of COLING 2002, the 19<sup>th</sup> International Conference on Computational Linguistics, pp. 516-522.
- Neema, S., Bapty, T., Gray, J., Gokhale, A., 2002, "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems". Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generators and Components (GCSE/SAIG), pp. 236-251.
- Nordstrom, G., Sztipanovits, J., Karsai, G., Ledeczi, A., 1999, "Metamodeling – Rapid Design and Evolution of Domain-Specific Modeling Environments". Proceedings of IEEE Conference and Workshop on Engineering of Computer-Based Systems, <http://www.computer.org/proceedings/ecbs/0028/00280075abs.htm>.

Pal, P., Loyall, J., and Schantz, R., et al., 2000, "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration". Proceedings of 3<sup>rd</sup> IEEE International Symposium on Object-Oriented Real-time Distributed Computing, <http://www.computer.org/proceedings/isorc/0607/06070310abs.htm>.

Raje, R. R., 2000, "UMM: Unified Meta-Object Model for Open Distributed Systems," Proceedings of ICA3PP'2000, 4<sup>th</sup> IEEE International Conference on Algorithms and Architecture for Parallel Processing, pp. 454-465.

Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., and Burt, C. C., 2001, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components". Proceedings of the 2001 Monterey Workshop on Engineering Automation for Software Intensive System Integration, pp. 109-119.

Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., and Burt, C. C., 2002, "A Quality of Service-based Framework for Creating Distributed Heterogeneous Software Components". Concurrency and Computations: Practice and Experience, vol. 14, pp. 1009-1034.

Rosa, N. S., Cunha, P., and Juso, G., 2002, "ProcessNFL: A Language for Describing Non-Functional Properties". Proceedings of 35<sup>th</sup> Hawaii International Conference on System Sciences, [http://www.hicss.hawaii.edu/HICSS\\_35/HICSSpapers/PDFdocuments/STDSL06.pdf](http://www.hicss.hawaii.edu/HICSS_35/HICSSpapers/PDFdocuments/STDSL06.pdf).

UML, 2002, "UML<sup>TM</sup> Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Revised Submission", <http://www.omg.org/cgi-bin/doc?realtime/03-05-02>.

van Wijngaarden, A., 1974, "Revised Report on the Algorithmic Language ALGOL 68". Acta Informatica, vol. 5, pp. 1-236.

Yang, C., Lee, B-S, Bryant, B. R., Burt, C. C., Raje, R. R., Olson, A. M., and Auguston, M., 2002, "Formal Specification of Non-Functional Aspects in Two-Level Grammar". Proceedings of the UML 2002 Workshop on Component-Based Software Engineering and Modeling Non-Functional Aspects (SIVOES-MONA), <http://www-verimag.imag.fr/SIVOES-MONA/uniframe.pdf>.