Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models[*]

Carol C. Burt
Barrett R. Bryant
University of Alabama Birmingham
cburt, bryant@cis.uab.edu

Rajeev R. Raje
Andrew Olson
Indiana University Purdue University Indianapolis
rraje,aolson@cs.iupui.edu

Mikhail Auguston
New Mexico State University
mikau@cs.nmsu.edu

Abstract

The UniFrame research project is proposing a Unified Component Meta Model Framework (UniFrame) that includes Quality of Service (QoS) contracts. Today it is the role of the software architect, based on experience, to design platform specific solutions that will meet QoS requirements. As we refine algorithms for model transformations, we must identify these QoS-aware design patterns and utilize them during model transformations. Our research includes supporting and participating in the exploration of generative techniques as they relate to QoS requirements (both static and dynamic) and the standardization of QoS-aware transformations. This paper explores how QoS requirements can impact decisions related to model transformation (using UML for Platform Independent Modeling and ISO IDL for the Platform Specific Model). It explores a series of QoS related design issues that must be considered as platform independent models are refined for specific component platforms.

1. Introduction

Enterprises are increasingly dependent upon multiple middleware technologies that enable new business paradigms by weaving together legacy systems with advanced technology. This technology supports core business functionality, enables distributed business systems, integrates business processes and enables companies to communicate with customers, suppliers, and business partners. While it is possible to construct heterogeneous component systems, it requires that the developer be aware of the nuances of the diverse middleware technologies. In addition, the increased complexity of this environment makes it impossible to predict the non-functional aspects of such a system until after it is constructed. That is, metrics and test scenarios must be hand crafted on a case-by-case basis to determine if a composition is acceptable. These problems must be resolved for the promise of software component technology to be fully realized.

The Unified Component Meta-Model Framework (UniFrame) [1] research is an attempt to unify distributed component models under a common meta-model for the purpose of enabling the discovery, interoperability, and collaboration of components via generative software techniques. This research targets the dynamic assembly of distributed software systems from components under different component models, and explores how the quality of service (QoS) requirements influences the design of components and their compositions.

Today, software architects leverage their experience in designing distributed systems when refining business and information technology models to ensure the quality of service requirements are met. To enable the use of generative techniques as models are refined, these experience-based design patterns must be formalized. As a part of this research, we plan to document the effect of design decisions on attaining quality of service requirements and explore techniques for providing the instrumentation necessary to measure QoS features. In this research we are focusing on two key QoS aspects for distributed component solutions: the security access control and the performance.

This paper explores the experience-based design considerations related to quality of service requirements during the model transformations when the Model Driven Architecture [2] techniques are used. It expands on previous work [3] that identified standards that are in progress as well as additional standards that are needed for the definition of QoS-based service contracts. For illustrative purposes, it presents design considerations

[*] This research was supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

for the security and the performance during the transformation of a simple Platform Independent Model (described in UML) to a CORBA model (described in ISO IDL). The future goals of our research include the identification and standardization of metrics necessary to validate the patterns and a mechanism to allow QoS related design patterns to be expressed as model parameters.

2. Model Driven Architecture

Model driven architecture techniques are not new; business and process modeling have been used for many years to capture requirements of information systems. As object-oriented analysis and design techniques matured, the Unified Modeling Language (UML) was standardized by OMG and became a popular technique for expressing both domain/business models and models of information systems.

OMG's Model Driven Architecture (MDA) [2] initiative facilitates the standardization of Platform Independent Model (PIMs) and the transformation of those models to multiple Platform Specific Models for implementation (such as CORBA, J2EE, or Web Services). In this way a single PIM can be used as the basis for multiple implementation technologies, and with standardization of the transformation algorithms, appropriate bridges can be generated. Standardizing platform independent models is a natural extension of existing OMG analysis and design standards for modeling and meta-modeling services. Standardizing multiple transformations to diverse technology platforms is a natural extension of the OMG mission to define interoperability standards.

Many OMG standards contain UML models to describe the domain model and/or semantics of services. Typically these domain models (expressed or implied) are independent of the CORBA platform (evidenced by the fact that they have been leveraged for use in J2EE and other technology platforms). In the past, OMG has only standardized the transformations to CORBA specific model(s) expressed in ISO IDL; however, it is expected that many of the existing services will be standardized for alternative platform technologies.

This focus on the Model Driven Architecture is a catalyst for the consideration of the effects of Quality of Service (QoS) requirements on computing models. At present, we have a limited ability to express QoS requirements as model parameters and even less definition of the algorithmic requirements to satisfy specific quality of service demands. The Model Driven Architectural vision, which is consistent with those of this research, includes standards that enable the use of generative techniques for construction of interoperability bridges between platform technologies. While this vision is appealing, there is a great deal of research to be done before this is feasible. The problem lies not in determining a single transformation from a platform independent model to a platform specific model, but in understanding the appropriate transformation based on quality of service requirements. Some of the model transformation issues related to the performance and security access control are discussed in this paper.

3. Relevant Standards and Known Issues

OMG has standardized technologies [18] that include a UML profile for CORBA and a UML profile for Enterprise Distributed Object Computing (EDOC). In addition, the Java Community Process has standardized a UML profile for Java2 Enterprise Edition (J2EE). These profiles, however, do not consider how to model QoS related aspects.

The OMG Meta-Object Facility provides a standard for generation of interfaces from MOF compliant UML models. However, it is well known that there are issues with this mapping for distributed solutions. The OMG Architecture board produced a paper that describes the technical details of the Model Driven Architecture (MDA) [3]. This document outlines areas where research is required before the MDA vision can be fully realized. The paper states: "It is generally agreed that the MOF-IDL mapping is in need of upgrading. The problem is that the generated interfaces are not efficient in distributed systems. Firstly, the mapping predates CORBA valuetypes and thus does not make use of them. Secondly, a class with N attributes is always mapped to a CORBA interface with N separate getter/setter operations. In a distributed system one would want to group attributes based upon use cases, cache attribute values, or implement other optimizations to reduce the number of distributed calls. Realistically we will probably have to accept the fact that for the foreseeable future, the automatically generated transformation from PIM to PSM will have to be enhanced by humans. As we gain more experience we will be able to define various patterns and allow them to be selected in some way."

In addition, security requirements often influence the technique utilized in transformation of a platform independent model to a platform specific model. It is widely accepted within the Model Driven Architecture community that generated interfaces must be optimized using the quality of service and usage scenarios. This requires research on the appropriate techniques for integrating QoS into the generative programming model [4] is necessary before standards can be progressed in this area.

4. MDA and Quality of Service

Although QoS parameters and associated metrics have been widely used in networking, there is no standard vocabulary for discussing the QoS as it relates to distributed computing and component-based solutions. For example, the CORBA® Components Specification only uses the term "quality of service" with regard to events and whether or not they are transactional in nature [5]. The Java2 Enterprise Edition (J2EE) specification [6] clearly states the expectation that J2EE products will vary widely and compete vigorously on various aspects of quality of service. Such products will provide different levels of performance, scalability, robustness, availability, and security, although in some cases the specification requires minimal levels of service.

A standard vocabulary is the first step toward progressing Model Driven Architectures that include QoS parameterization and/or QoS contracts. This is one of the goals of the UniFrame research.

4.1 Previous and Related Work

As a part of the UniFrame research, we have outlined an approach to a QoS-based framework for creating distributed heterogeneous software components [7]. The QoS-based method in UniFrame is made up of three steps:

1. The creation of a catalog for QoS parameters (or metrics),
2. A formal specification of these parameters, and
3. A mechanism for ensuring these parameters, both at each individual component level and at the entire system level.

Our work leverages the research work by Zinky, Bakken and Schantz [8] with a goal of providing a catalog of QoS parameters and indicating how parameters might be described. There are many possible QoS parameters that a component (and its developer) can use to indicate the associated service. Some of these parameters may be general in nature, while others may pertain to a specific domain. The goal of creating the QoS catalog is two fold: a) it assists the component developer (or the system integrator) in selecting the necessary QoS parameters for the component (or system) under construction, and b) it enables the developer (or integrator) to ensure the necessary QoS guarantees by integrating the selected QoS parameters into the assurance process. We have created a preliminary version of the QoS catalog in [15]. In addition to identifying and describing different QoS parameters, this catalog also classifies them and provides models for their compositions.

Other relevant research work in this area includes Frolund and Koistinen [9] who point out that deciding which quality of service properties should be provided by individual components is an important part of the design process. They define a Quality-of-Service specification language (QML) and they show how the Unified Modeling Language (UML) can be extended to support the concepts of QML. They also show how to represent QML constructs in terms of ISO Interface Definition Language (IDL) [9] [10]. There are also case studies where Object Constraint Language (OCL) is being used a mechanism for the annotation of UML models for the purpose of expressing security constraints [11]. Recent work in adaptive systems extends the work in Quality Objects (QuO) [12] with security specific strategies that use the QuO contract definition language (QDL) [13].

We expect standards activity in this area will consider and leverage the experience and results of these efforts.

4.2 Recent Standards Activity

In January 2002, the OMG Analysis and Design task force issued a RFP (Request for Proposals) for a "UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms" [14]. This RFP solicits proposals for a UML profile or Meta Object Facility (MOF) meta-model that defines standard paradigms of use in modeling quality of service and fault-tolerance aspects of systems. This is the first of a series of RFPs that have the goal of significant benefits to the UML user community engaged in high-quality robust system development. The mandatory requirements of this RFP are listed in Figure 1.

As distributed systems are becoming more omni-present with many of them handling mission-critical applications, the notion of QoS-oriented software development is of paramount importance. Such a quality-oriented approach, in addition to providing seamless access to heterogeneous components, will also ensure the reliability and a high confidence of

1. A General Quality of Service Framework

To ensure consistency in modeling various qualities of service, submissions shall define a standard framework or, reference model, for QoS modeling in the context of the UML. This shall include:

- A general categorization of different kinds of QoS; including QoS that are fixed at design time as well as ones that are managed dynamically
- Integration of different categories of QoS for the purpose of QoS modeling of system aspects.
- A coherent set of stereotypes, tagged values, and constraints as necessary to represent the identified QoS properties constructing a UML Profile.
- Identification of the basic conceptual elements involved in QoS and their mutual relationships. This shall include the ability to associate QoS characteristics to model elements (specification), a generic model of the system aspects involved in QoS-associated collaboration and their functional interactions and use cases (usage model), and a generic model of how QoS allocation and decomposition is managed.

2. A Definition of Individual QoS Characteristics

Submissions shall define QoS characteristics, particularly those important to real-time and high confidence systems, which describe the fundamental aspects of the various specific kinds of QoS based on the QoS categorization identified in the framework. These shall include but are not limited to the following:

- time-related characteristics (delays, freshness)
- importance-related characteristics (priority, precedence)
- capacity-related characteristics (throughput, capacity)
- integrity related characteristics (accuracy)
- fault tolerance characteristics (mean-time between failures, mean-time to repair, number of replicas)

3. A coherent set of stereotypes, tagged values, and constraints as necessary to represent the identified QoS properties constructing a UML Profile.

Figure 1: OMG RFP UML Profile for QoS Mandatory Requirements

distributed systems software. As indicated earlier, the need for standardization of a quality of service vocabulary was recognized early in our research and we are carefully tracking the work of the OMG in this area as we continue to progress our work in the development of a quality of service catalog [15].

5. Models Transformations

UML is a graphical notation for expressing models; it is important to understand that many alternative modeling syntax exist – for example, the XML Model Interchange (XMI) format leverages Extended Mark-up Language (XML) to express Meta-Object Facility (MOF) compliant models. While there is a standard UML profile for CORBA, the ISO IDL continues to be the most common notation used to define a CORBA model. Our research is also exploring the use of two-level grammar (TLG) as a formal mechanism for expressing models [16]. These text notations are useful for computers as they process textural or binary syntax more efficiently than graphics. Mappings from one notation to another are often produced and used for various analysis tasks (sometimes preserving all model information, and sometimes losing information which has no equivalent in an alternative modeling syntax). For example, IDL models may be expressed in the UML profile for CORBA. Such mappings are not transformations – they are merely alternative representations of the same model.

A model transformation occurs when models are refined and details are added for the purpose of focusing on a particular implementation technology or an aspect of the domain model. Model transformations are used to document different "levels of abstractions", "viewpoints" or "aspects" of an information system. Models that comply to a specific meta-model may utilize generative techniques for the transformations; leveraging information that the generator knows regarding the target implementation platform and/or parameterizations provided by the software architect. To fully realize the potential of the MDA, the Quality of Service (QoS) catalogs, the formal parameterization of Platform Independent Models, and ultimately the instrumentation generation rules must be standardized within the Model Driven Architecture roadmap.

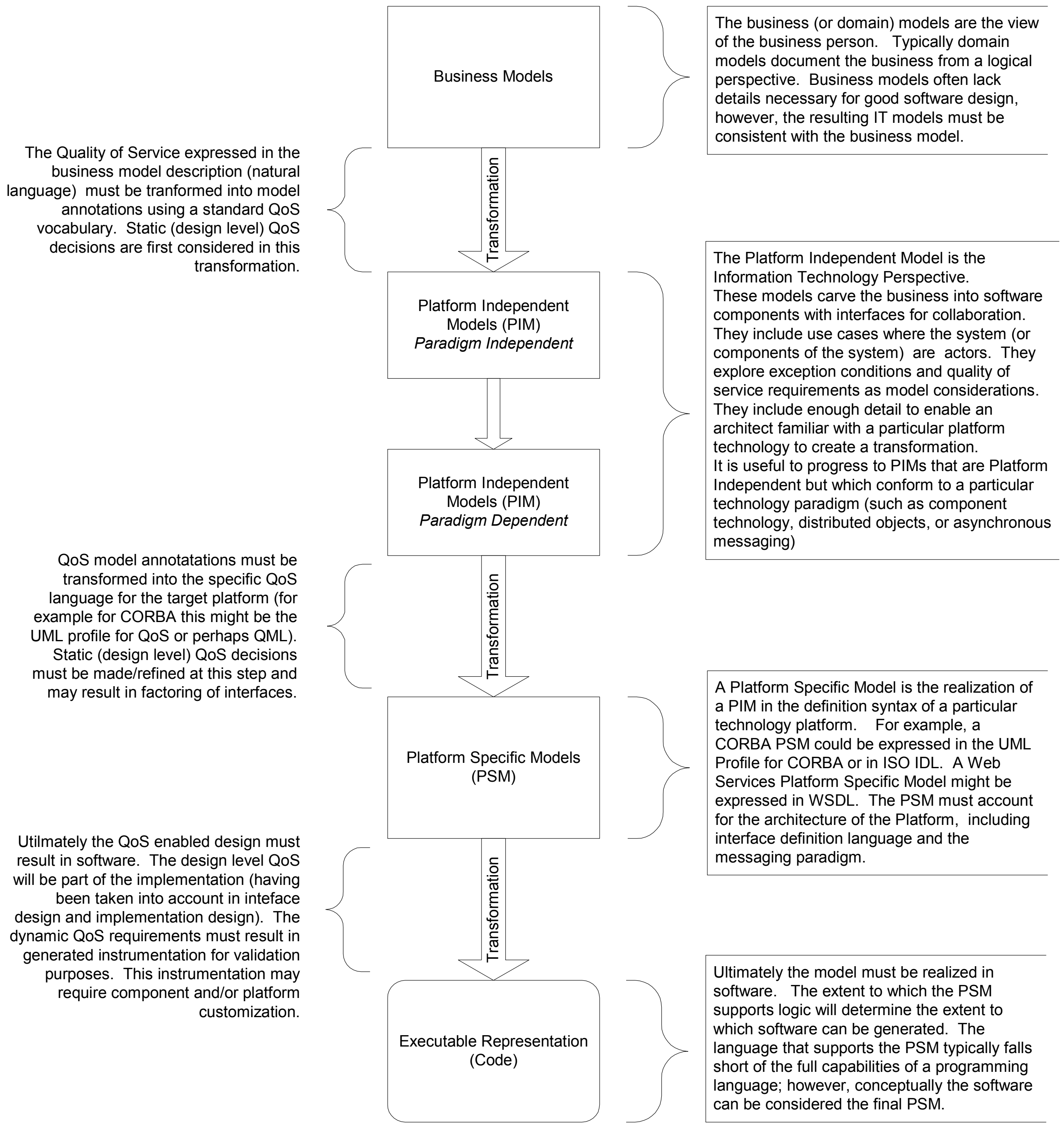
Business Models
Transformation
Platform Independent Models (PIM) *Paradigm Independent*
Platform Independent Models (PIM) *Paradigm Dependent*
Transformation
Platform Specific Models (PSM)
Transformation
Executable Representation (Code)
The business (or domain) models are the view of the business person. Typically domain models document the business from a logical perspective. Business models often lack details necessary for good software design, however, the resulting IT models must be consistent with the business model.
The Quality of Service expressed in the business model description (natural language) must be tranformed into model annotations using a standard QoS vocabulary. Static (design level) QoS decisions are first considered in this transformation.
The Platform Independent Model is the Information Technology Perspective. These models carve the business into software components with interfaces for collaboration. They include use cases where the system (or components of the system) are actors. They explore exception conditions and quality of service requirements as model considerations. They include enough detail to enable an architect familiar with a particular platform technology to create a transformation. It is useful to progress to PIMs that are Platform Independent but which conform to a particular technology paradigm (such as component technology, distributed objects, or asynchronous messaging)
QoS model annotatations must be transformed into the specific QoS language for the target platform (for example for CORBA this might be the UML profile for QoS or perhaps QML). Static (design level) QoS decisions must be made/refined at this step and may result in factoring of interfaces.
A Platform Specific Model is the realization of a PIM in the definition syntax of a particular technology platform. For example, a CORBA PSM could be expressed in the UML Profile for CORBA or in ISO IDL. A Web Services Platform Specific Model might be expressed in WSDL. The PSM must account for the architecture of the Platform, including interface definition language and the messaging paradigm.
Utilmately the QoS enabled design must result in software. The design level QoS will be part of the implementation (having been taken into account in inteface design and implementation design). The dynamic QoS requirements must result in generated instrumentation for validation purposes. This instrumentation may require component and/or platform customization.
Ultimately the model must be realized in software. The extent to which the PSM supports logic will determine the extent to which software can be generated. The language that supports the PSM typically falls short of the full capabilities of a programming language; however, conceptually the software can be considered the final PSM.

Figure 2 – QoS considerations during model transformation

Figure 2 outlines the models that are commonly progressed in a MDA approach. Quality of Service annotations or parameters must be introduced into each model and the transformations must consider such parameters as models are refined. The current OMG RFP is a beginning – standardizing a vocabulary and syntax for expressing QoS in UML. As we move beyond the QoS catalog, our research will focus on the constraints that are placed on transformations as a result of the quality requirements and explore generative techniques for ensuring that metrics can be gathered. In addition, use case scenarios must be formally expressed so that they can be used as an input to an interface generator. Thus, an ultimate goal is that given a parameterized domain model, semantically equivalent interfaces (and the bridges between them) might be generated. Our future work will explore mechanisms for expressing such parameters as annotations for design patterns so that this vision can be progressed.

In the example described below, we will follow the progression of a business model for a simple bank to a CORBA Platform Specific Model that uses experienced-based design patterns to address Quality of Service requirements. We will look at how these patterns allow security administration to be simplified and the most common remote services to be optimized. The final set of interfaces will be presented as the "UniFrameBank".

6. An Example: Model Driven Architecture with Quality of Service Considerations

Model Driven Architecture starts with the construction of a business (or domain) model based on the requirements analysis. Requirements are often expressed in a natural language and UML is a popular tool for documenting and validating the business model. In this example, we will analyze a "simple bank" and explore how interfaces may be organized based on the quality of service aspects and known use case scenarios.

6.1 Simple Bank Business Model

A typical business description of a simple bank is: The SimpleBank manages accounts. A unique account number identifies each account. An account has items associated with it. An item is a transaction against the account (deposit, withdrawal or adjustment). Deposits and withdrawals have a unique identifier, a date and an amount. Adjustments have these attributes and an annotation that provides the reason for the adjustment. There is a bank identifier or bank routing number that is used as an account prefix when interfacing with other banks. This bank id is not, however, used internally as part of the account number. Accounts maintain an owner identifier, a single PIN number, and an available balance. The SimpleBank supports the opening and closing of accounts and update of account information such as owner and PIN. Accounts are typically located using the account number, but can also be located using the owner identifier. Some business services require that the PIN be validated before the transaction can be completed.

Figure 3, based on the above description, indicates the UML business model for the simple bank.

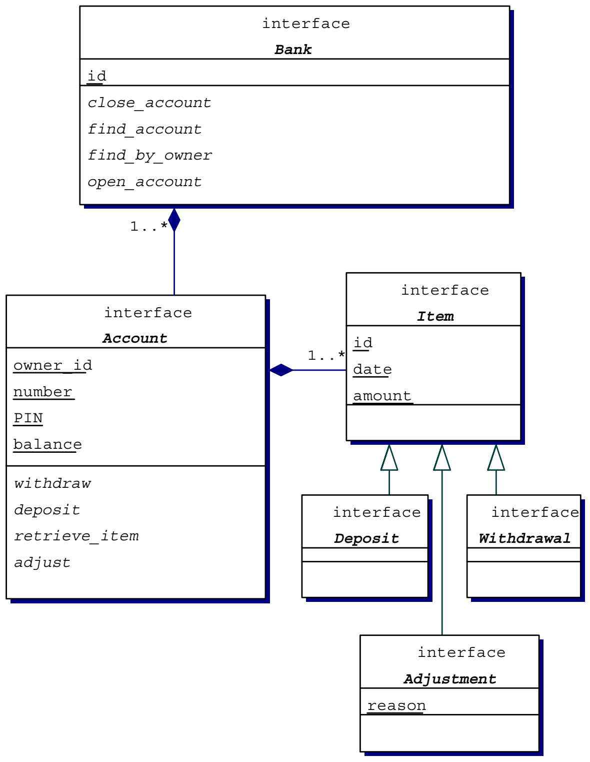

Figure 3: The simple bank business model

6.2 Simple Bank Platform Independent Model

The next step is to determine the usage scenarios that must be supported by our SimpleBank, to fully explore the business rules and to determine the quality of service characteristics of the usage scenarios (or services). This is necessary to create a Platform Independent Information Technology (IT) model of the SimpleBank that enables efficient information technology services to be offered by the SimpleBank. We need to resolve questions that arise during the development of the business model such as:

- Can one owner have multiple accounts?
- If one owner can have multiple accounts, how do we navigate to them?
- Is there a need to iterate through account items?
- What are the most common usage scenarios?
- How do we optimize the services to accommodate the common usage patterns?

A use case analysis is employed to capture this information. The Platform Independent Model considers additional details such as exceptions and security considerations that are not unique to a particular platform. Abstracting away such details is typical of business models, but those issues must be considered for an information technology system. A common initial approach to defining the Platform Independent Model is to add design details directly to the business model. This is typically not sufficient as business models are often not appropriate for expressing information technology viewpoints. For this reason, a software architect, drawing on their own experienced-based design patterns and taking all aspects of the model into consideration, transforms the business model into a PIM. This paper discusses the transformation of model and outlines some of these experienced-based techniques. It is hoped that these experience-based techniques will be formalized in the future for the purpose of using them with generative algorithms.

During use case analysis for the SimpleBank, we capture the following business rules that must be supported by the information system (this is a subset provided to aid in illustration of the QoS requirements).

- Bank customers may query account balance (via phone) and/or withdraw funds (using a teller machine) from an account without assistance provided that they have their account number and PIN.
- Merchants may request withdrawals from accounts by providing their merchant identification, account number and PIN (checkcard services).
- Tellers may locate accounts based on owner identification, query account balances, process deposits and withdrawals for customer and review existing account items. Tellers may use external means of identifying a customer (not required to use/know PIN).
- Bank managers may perform all the functions of a Teller and may also open and close accounts and create adjustments.
- Bank customers may have many accounts and will use the same owner identifier for all these accounts. It must be easy to locate all the accounts for a customer.
- The bank offers a response time guarantee of three second to merchants for services or the fees are waived for the request. Merchant requests must be prioritized above other system requests. Response times for merchant requests must be monitored.
- Account balance inquires from remote locations are a very common business scenario that requires less than five second response time to ensure customer satisfaction. Response time on balance inquires must be monitored.

The first quality of service issue we will address is one aspect of security: access control. We will use the techniques outlined in Figure 4 to review the model and use cases and apply experience-based security access control design patterns.

Are there significant security requirements identified for the service(s)?

If so, consider segregating administrative features into separate interfaces from those that provide the less restrictive non-administrative functionality

Is it expected that administrators will also be allowed to use all the non-administrative features of a service?

Use inheritance to clarify this in the model and simplify the security model. That is, an administrative interface should inherit from the non-administrative interface.

Can you navigate between interfaces as required while maintaining security controls at the point of navigation?

Review navigation patterns to ensure that given an object reference, it will be easy to navigate to other objects and that security rules logically apply at the point of navigation.

Figure 4: Experience-based Security Techniques

It is much easier for security administrators to assign policies based on roles to groups of functionality (vs. individual users and individual functions). If functionality can be grouped based on security patterns (such as view access vs. administration access) then security policy can be defined based on functional groupings (ultimately interfaces and/or objects). This also increases the scalability of the security model and is more efficient at run-time.

Our analysis review indicates that there are significant security related usage restrictions, and that using the business model as the basis of the PIM without refinement for security considerations would force access control checks for each individual operation. For example, our business rules state that the open_account() and close_account() operations can only be done by a bank manager, but they are in the same interface as the find_account() operation that locates accounts and must be accessible to tellers. In addition, we see that bank managers are allowed to perform all the functions of tellers, so we can use inheritance to capture this aspect of the access requirements. Finally, we need to review the navigation patterns to ensure that our Platform Independent Model supports all our usage scenarios.

During our analysis, we notice that we have two interfaces that are empty – that is, they provide no additional functionality (other than typing). We may want to simplify this in the IT model. A refined Platform Independent Model (created based on the above discussion) is shown in Figure 5.

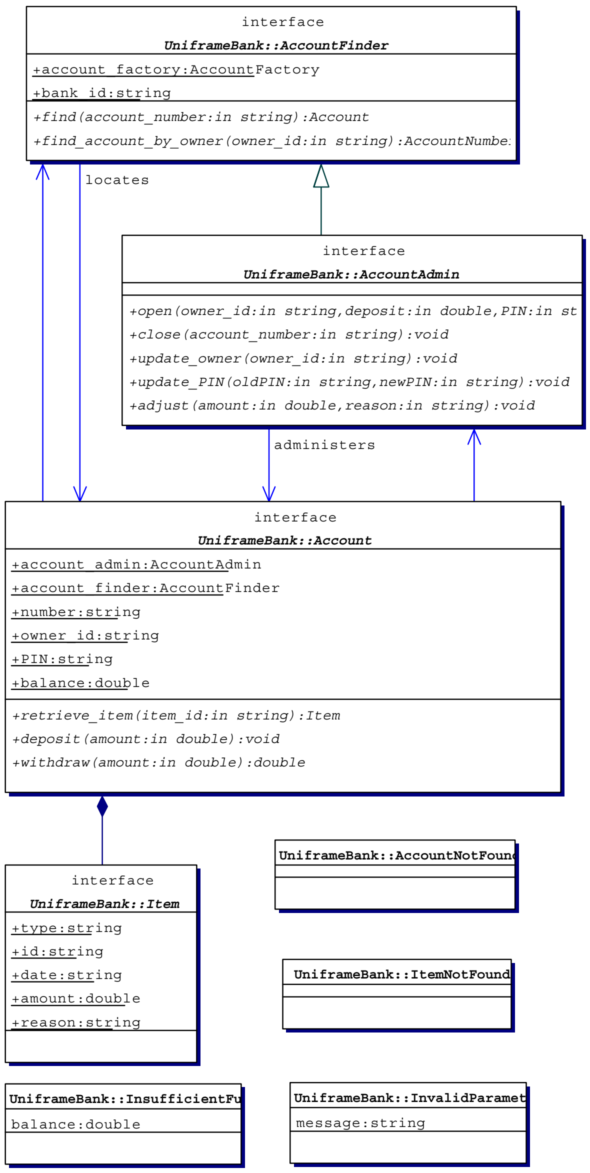
interface
UniframeBank::AccountFinder
+account_factory:AccountFactory
+bank_id:string
+find(account_number:in string):Account
+find_account_by_owner(owner_id:in string):AccountNumbe
locates
interface
UniframeBank::AccountAdmin
+open(owner_id:in string,deposit:in double,PIN:in st
+close(account_number:in string):void
+update_owner(owner_id:in string):void
+update_PIN(oldPIN:in string,newPIN:in string):void
+adjust(amount:in double,reason:in string):void
administers
interface
UniframeBank::Account
+account_admin:AccountAdmin
+account_finder:AccountFinder
+number:string
+owner_id:string
+PIN:string
+balance:double
+retrieve_item(item_id:in string):Item
+deposit(amount:in double):void
+withdraw(amount:in double):double
interface
UniframeBank::Item
+type:string
+id:string
+date:string
+amount:double
+reason:string
UniframeBank::AccountNotFoun
UniframeBank::ItemNotFound
UniframeBank::InsufficientFu
balance:double
UniframeBank::InvalidParamet
message:string

Figure 5: Simple Bank Platform Independent Model (PIM)

6.3 Simple Bank Paradigm Specific Model

The next step in the Model Driven Architecture is to find a way to use **all** the model information that has been captured in the use case analysis of the Platform Independent Model (PIM) and define the techniques that allow Platform Specific Models to be created that leverage all aspects of the PIM. We have reached the point where model optimizations must consider the characteristics of the target environment and/or platform.

As we make this transition, we see the value of progressing to a Platform Independent Model that is optimized for a particular computing paradigm; that is, the PIM may be used as a foundation for multiple platform specific implementations provided those platforms share some common characteristics. The characteristics or paradigms to consider include distributed solutions (distributed objects, synchronous messaging, asynchronous messaging, etc.), and local solutions (object-oriented programming, procedural programming, etc.). Other aspects such as embedded and/or real-time might also be considered at this time. A transition to a "paradigm specific model" is a useful intermediate step that captures the analysis necessary for a transition from a Platform Independent to Platform Specific Models. As such it may be useful in the development of algorithms that can be used with generative techniques for Platform Specific Models.

Are there usage scenarios that require remote access across wide area networks where network speed may be a factor?

Evaluate carefully each high usage remote access scenario for the following characteristics.

1. Does it require multiple network operations to accomplish what is logically a single request to the user?

Consider creating a service interface that offers services that wrap the existing service and gather all required information before responding.

2. Is it common to require and/or update multiple attributes simultaneously?

Consider passing structures or objects by value instead of using accessors and mutators on object attributes.

Figure 6: Experience-based remote access techniques

Continuing with our evaluation of quality of service issues, we focus on a distributed object paradigm as our technology choice. Figure 6 includes some of the techniques that we can use to refine our Platform Independent Model (these are illustrative and not exhaustive).

These design principles are key features of aspect oriented (or service oriented) architectures and are at the heart of what must be done for secure manageable web services. It is important to note that the business model is typically expressed as an object-oriented view of the business, not as a service oriented model. Therefore it is not possible to derive the service model directly from a business model with generative tools – that is, there is additional information (such as patterns of usage) that is not expressed in the business model that must be considered. The effect of this is that the resulting service model must be manually validated against the business model, as effects of changes on one model are not readily identifiable. This is a serious issue for business systems and one that will need to be addressed as MDA techniques and tools mature.

The analysis of our SimpleBank indicated that remote requests for account balance are very common and have a performance commitment associated with them. In addition, there were merchant services that have an impact on the revenue if performance commitments are not met. The PIM currently requires two independent requests across the network each time a balance is requested – first AccountFinder::find (account_number) to locate the account followed by Account::balance() to retrieve the balance. A more efficient remote operation (on some yet to be determined interface) might be get_balance (account_number) to allow this to be a single remote operation.

The key services of the bank are reflected in Figure 7 This is a paradigm specific model that is an addendum to the PIM. This reflects the requirement that distribution be considered and that key services be segmented for the purpose of performance enhancement, prioritization, and metrics.

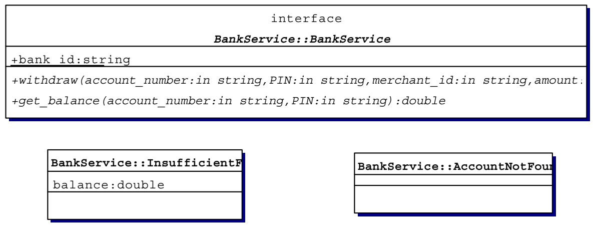
interface
BankService::BankService
+bank_id:string
+withdraw(account_number:in string,PIN:in string,merchant_id:in string,amount:
+get_balance(account_number:in string,PIN:in string):double
BankService::InsufficientF
balance:double
BankService::AccountNotFou

Figure 7: Key Services Model for the SimpleBank

6.4 Simple Bank – Platform Specific Model

As we consider the technology platform that will be utilized, an evaluation of the quality of service requirements for the Simple Bank with regard to the platform features are part of the final transformation into a Platform Specific Model. These requirements (based on the analysis of the domain and the QoS parameters from our catalog) are:

Security: the service should be able to support dynamic decisions regarding exporting functionality to a user. The user should not be aware or have the ability to attempt to invoke any update operations unless they are authorized for update (that is, it should be possible at runtime to determine the interface offered to individual users).

Capacity: the system should be architected to scale to thousands of uses doing concurrent extensive work on hundreds of accounts. The usage is such that multiple operations will typically be done on accounts.

Maintainability: The ability to provide administrative services that extend the functionality must be available. The enhancement of these administrative interfaces should not impact the customers who are using the core features of the bank.

Performance: This is a distributed service. The service should be optimized for interactions across a wide area network at midrange speed.

The UniFrameBank designed to accommodate these requirements is defined below in ISO IDL. Note once again, that this interface model cannot be generated from the business model; a classic object-oriented design that does not take into consideration any QoS characteristics. The level of abstraction for the business model does not support the level of detail required to factor functionality in this way.

The UniFrameBank module defined in Figure 8 takes these QoS requirements and the usage scenarios into account. It introduces interfaces that respect the QoS requirements of the SimpleBanks' service offerings while maintaining the separation of concerns necessary to address security, ease of administration and maintainability.

```
module UniframeBank {

    typedef sequence<string> AccountNumbers;

    struct AccountInfo {
        string owner_id;
        string number;
        string PIN;
        double balance;
    };

    struct ItemInfo {
        string id;
        string type;
        string date;
        double amount;
        string reason;
    };

    exception AccountNotFound{};
    exception ItemNotFound{};
    exception InsufficientFunds{
       double balance;
    };
    exception InvalidParameter{
string message;
    };

  // Forward references
  interface AccountFactory;
  interface Account;
  interface AccountAdmin;

  // Key Services
  interface BankService
  {
    readonly attribute string bank_id;

    void withdraw(
            in string account_number,
            in string PIN,
            in string merchant_id,
            in double amount
    ) raises (
            AccountNotFound,
            InsufficientFunds
    );

    double get_balance(
            in string account_number,
            in string PIN
    )
            raises (AccountNotFound
    );
  };

  // AccountFinder
  interface AccountFinder {
    readonly attribute
              AccountFactory account_factory;
    readonly attribute string bank_id;

    Account find(
        in string account_number
    ) raises (
        AccountNotFound
    );

    AccountNumbers find_account_by_owner(
        in string owner_id
    ) raises (
        AccountNotFound
    );
  };
```

```
  // AccountFactory
  interface AccountFactory : AccountFinder {

    AccountInfo open(
        in string owner_id,
        in double deposit,
        in string PIN
    ) raises (
        InvalidParameter
    );

    void close(
        in string account_number
    ) raises (
AccountNotFound
    );
  };

  // Account
  interface Account {

    readonly attribute
           AccountAdmin  account_admin;
    readonly attribute
           AccountFinder account_finder;
    readonly attribute string number;
    readonly attribute string owner_id;
    readonly attribute string PIN;

    ItemInfo retrieve_item_info(
         in string item_id
    ) raises (
        ItemNotFound
    );

    void deposit (
          in double amount
    ) raises (
          InvalidParameter
    );

    double withdraw (
        in double amount
    ) raises (
        InsufficientFunds
       );
  };

  // AccountAdmin
  interface AccountAdmin :  Account   {
    void update_owner(
         in string owner_id
    ) raises (
         InvalidParameter
    );

    void update_PIN(
          in string oldPIN,
          in string newPIN
    ) raises (
          InvalidParameter
    );

    void adjust(
         in double amount,
         in string reason
    ) raises (
          InvalidParameter
    );
  };
};
```

Figure 8 – UniFrameBank – Platform Specific Model

An AccountFinder interface is responsible for locating accounts. This eases security because none of the operations on the Account are visible from this interface; hence if a client is not authorized to access an account they will be restricted from obtaining a reference to an Account object. In addition, the AccountFactory (which inherits from AccountFinder) is available only to clients who are authorized to open or close accounts. The AccountAdmin interface was introduced to allow evolution to more sophisticated services without affecting the interfaces of the coreaccount services (AccountFinder and AccountFactory). The client must be authorized to use an AccountAdmin object which had the ability to modify existing account attributes or items. The Account object offers only the core banking operations. The ability to request all Account or Item information in a single operation was added to the Account Interface to meet the performance requirements and limit the number of network interactions. The BankService interface that was introduced in the paradigm specific PIM is retained in the PSM.

7. Future Directions

The models and IDL presented in this paper will form the basis of the additional work to validate the experience-based design patterns presented in the paper and to progress techniques for the model parameterization with the goal of enabling generation of platform specific models such as those presented in this paper.

The next step in our research is to examine how these experienced-based patterns can be expressed as model parameters. We are hopeful that previous research (including our work on the QoS Catalog and TLG) [15] [16] and work in progress on standards for UML Profiles for QoS [14] can be leveraged. For this reason, we have not proposed a language for this purpose as yet. The QoS instrumentation is a complementary research activity. There is a need in component-based environments to progress instrumentation that can be utilized to determine whether a component can meet those QoS parameters when used within a composition. Of course, a part of the challenge is that the instrumentation introduces an additional overhead and in situations that are time sensitive or must be predictable, this overhead may disrupt the ability to measure the QoS parameter under observation. It is clear that a substantial amount of research needs to be done in this area and we plan to use an approach based on event grammars as indicated in [1] [17].

8. Conclusion

The ability to provide the QoS parameterization of models is recognized in the Object Management Group community and standards in this area will lead to the ability to generate Platform Specific Models that take quality of service characteristics into account. However, since there has been a very little work on progressing Quality of Service specifications for component-based architectures, UniFrame research has a potential to impact how the Object Management Group (OMG) defines QoS parameterization for Model Driven Architecture and the ability to more clearly specify and measure component feasibility for a particular task. The standardization of QoS catalogs and parameters is a pre-requisite to defining algorithms for the transformation of Platform Independent Models into Platform Specific Models. In addition, benchmarking and service validation via instrumentation require that such standards exist. Our expectation is that any Quality of Service parameters defined by OMG will be applicable for CORBA®, J2EE™, and Web Services component architectures.

Quality of Service characteristics must have syntax for expression in every artifact of the analysis, design and development process. Design patterns must be documented and exploited in such a way that generative techniques can be applied. In addition, formal specifications will allow the instrumentation necessary for measuring quality of service to be come an integral part of middleware and component implementation frameworks.

QoS-oriented software development is of paramount importance to delivering robust, scalable and secure distributed component solutions.

9. References

[1] Rajeev R. Raje, Barrett Bryant, Mikhail Auguston, Andrew Olson, Carol Burt. "*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, Monterey, California, 2001.

[2] Object Management Group. 2001. *Model Driven Architecture: A Technical Perspective*. Technical Report. Document # ormsc/2001-07-01. Framingham, MA: Object Management Group. July 2001.

[3] Carol C. Burt, Barrett R. Bryant, Rajeev R. Raje, Andrew Olson. Mikhail Auguston. 2002. *Quality of Service (QoS) Standards for Model Driven Architecture.* Proceedings of the 2002 Southeastern Software Engineering Conference (to appear).

[4] K. Czarnecki, U. W. Eisenecker, 2000. *Generative Programming: Methods, Tools, and Applications.* Addison-Wesley.

[5] Object Management Group. 2001. *CORBA 3.0 CORBA Component Model Chapters*. Document # ptc/2001-11-03. Framingham, MA: Object Management Group.

[6] Sun Microsystems. 2001. *Java 2 Platform Enterprise Edition Specification v1.3*, Available via ftp from www.java.sun.com. Sun Microsystems.

[7] Rajeev R. Raje, Mikhail Auguston, Barrett Bryant, Andrew Olson, Carol Burt. 2001. *A Quality of Service-based Framework for Creating Distributed Heterogeneous Software Components*. Technical Report. Indiana University Purdue University Indianapolis.

[8] J. A. Zinky, D. E. Bakken, R. Schantz,, 1995. *Overview of Quality of Service for Distributed Objects*, Proceedings of the Fifth IEEE Dual Use Conference.

[9] S. Frolund, J. Koistinen. 1998. *Quality of Service specification in Distributed Object Systems*, Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '98).

[10] S. Frolund, J. Koistinen. 1999. *Quality of Service Aware Distributed Object Systems.* 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '99).

[11] Ringo Ling, Hugo Latapie, Vu Tran, 2002. *Expressing Common Criteria Security Requirements in Domain Models in Model-base Architecture.* Technical Presentation. Distributed Object Security Conference (DocSec 2002). Baltimore, MD. March 2002.

[12] BBN Corporation, 2001. *Quality Objects (QuO) Project*, URL: http://www.dist-systems.bbn.com/tech/QuO.

[13] Chris Jones, Partha Pal, Franklin Webber, 2002. *Defense Enabling Using QuO: Experience in Building Survivable CORBA Applications.* Technical Presentation. Distributed Object Security Conference (DocSec 2002). Baltimore, MD. March 2002.

[14] Object Management Group. 2002. *UML™ Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms.* Request for Proposal. OMG document ad/02-01-07. Framington, MA. Note: This RFP issued January 2002 with submissions due June 24, 2002.

[15] Girish J. Brahnmath, Rajeev R. Raje, Andrew M. Olson, Mikhail Auguston, Barrett R. Bryant, Carol C. Burt. 2002. *A Quality of Service Catalog for Software Components*. Proceedings of the 2002 Southeastern Software Engineering Conference (to appear).

[16] Barrett Bryant, Mikhail Auguston, Rajeev R. Raje, Andrew M. Olson, Carol C. Burt. 2002. *Formal Specification of Generative Component Assembly using Two-Level Grammar*. Technical Report. University of Alabama Birmingham.

[17] Mikhail Auguston. 2000. *Tools for Program Dynamic Analysis, Testing, and Debugging Based on Event Grammars*. Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE 2000), pp.159-166.

[18] Object Management Group. 2000-2002.*OMG Adopted Technology for UML, UML Profiles, Meta Object Facility and Common Meta-Data Warehouse.* These OMG documents are available from OMG via http://www.omg.org/technology/documents/modeling_spec_catalog.htm. Framingham, MA: Object Management Group.