Integrated Design and Process Technology, **IDPT-2003**
Printed in the United States of America, June, 2003

MODELING WEB SERVICES: TOWARD SYSTEM INTEGRATION IN UNIFRAME

Fei Cao, Barrett R. Bryant, Carol C. Burt, Jeffrey G. Gray
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294, USA
{caof, bryant, cburt, gray}@cis.uab.edu

Rajeev R. Raje, Andrew M. Olson
Department of Computer and Information Science
Indiana University Purdue University at Indianapolis
{rraje, aolson}@cs.iupui.edu

Mikhail Auguston
Computer Science Department
Naval Postgraduate School
auguston@cs.nps.navy.mil

ABSTRACT

Web Services offer a platform independent solution for system integration in a distributed environment. But Web Services are weak in representing the business semantics of application domains. This paper presents a model-driven approach for specifying domain-specific component models in an effort to complement the current Web Services technology in terms of enriching the semantics representation. Web Services Description Language (WSDL) can then be generated automatically from the models with generators. The modeling of domain-specific components serves as a front-end to represent the semantics of components as well as for formalizing components while the generated artifacts facilitate component service synthesis.

1. Introduction

The integration and reuse of legacy software systems offer a promising direction for boosting productivity by dramatically reducing both cost and time-to-market expenses. One of the Object Management Group (OMG) initiatives is Model Driven Architecture (MDA)[1], in which legacy systems and Commercial-Off-The-Shelf (COTS) software can be transformed by reverse engineering into Platform Independent Models (PIMs) representing business functionality with underlying technical details presented abstractly. If this effort is successful, legacy systems and COTS software can be reintegrated into new platforms efficiently and cost-effectively. But for legacy systems and COTS software, the business logic and the software structures are usually encapsulated as black boxes, which makes it difficult to be reverse engineered. Hence, it is necessary to include the design artifacts (such as models, high-level specifications, etc.) in the business components. To that end, the vision of MDA also includes packaging models together with parameterized generators. The application generator will produce customized components according to the configuration parameters. In that way, not only can the footprint of business systems be minimized, but also various kinds of artifacts of business system can be generated on demand for system synthesis.

On the other hand, Web Services (WS)[2] technology offers a platform-independent solution for Enterprise Application Integration (EAI) by wrapping legacy systems as WS [Grah02]. Combining the model-driven approach with WS technology, software systems can be produced by synthesizing distributed models using generator technology.

UniFrame [Raje01] is a framework for seamless integration of heterogeneous distributed software components to assemble a complete distributed software system. The assembly process involves the generation of glue/wrapper code [Brya02], which is a challenging ad-hoc task considering the heterogeneous nature of distributed components. Because WS are based on open industry standards working across different platforms, wrapping heterogeneous components with WS for integration will transform the assembly task from n*m to n*1 processes (see Figure 1). The contribution of this paper is to propose the use of WS as a potential vehicle for system integration in UniFrame by enhancing semantic expressive power of WS using the model-driven approach. The related process is described herein.

In this paper, we present an approach based upon the principle of Model-Integrated Computing (MIC) [Léde01] to model the business domain-specific UMM component models. This involves a graphical modeling

[1] http://www.omg.org/mda/

[2] http://www.w3.org/2002/ws/

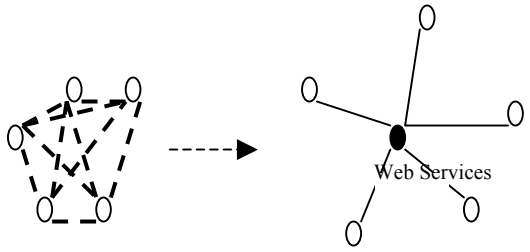


Figure 1. Reducing Gluing/Wrapping Process

environment for customizing a domain system based on domain-specific meta-models. An interpreter is built to generate WS Description Language (WSDL)[1] for business service integration. A generator can also be created to directly synthesize the implementation code. Figure 2 gives an overview of the approach.

This paper is organized as follows. Section 2 introduces the background knowledge of the UniFrame project, for which the proper meta-model development is imperative. Section 3 introduces the modeling environment and modeling targets with regard to the UMM model and WS. Section 4 describes the interpreter that generates the WSDL. A banking example is given in Section 5 illustrating the proposed approach. This paper ends with the conclusions and outlook in Section 6.

2. UniFrame

UniFrame is based on the Unified Meta-Component Model (UMM) [Raje00] for describing components. Systems constructed by component composition should meet both functional and non-functional requirements such as Quality of Service (QoS) requirements [Raje02]. UniFrame includes a specification of appropriate QoS parameters, which provide metrics of service at both the component level and system level, so that the software system produced by assembling heterogeneous components can be benchmarked over not only functional requirements, but also non-functional criteria. A Generative Domain Model (GDM) [Czar00] is used to describe the properties of domain- specific components and to elicit the rules for component assembly.

2.1 UMM

In the Unified Meta-Component Model (UMM), we are concerned about the following three aspects:

a) Component:

In [Medv97], components are described as being composed of the following aspects: interface, types, semantics, constraints and evolutions. But, this view does not reflect the collaborative features of distributed components. We believe that a component, as a provider for computational functionality and a gateway for further resource offerings, has not only computational aspects, but also cooperative aspects in distributed environments, as well as other auxiliary aspects like mobility and security.

b) Service and Service Guarantees.

Here we are focusing on providing metrics for quantifying the services provided by components as a criteria for making choices from multiple service providers, as well as criteria of judging assembled system by composing components. Once a component does not satisfy the expected QoS, it is a candidate for substitution. By modeling QoS aspects in the meta-model, we can weave the QoS instrumentation into generated code for QoS measurements at deployment time.

c) Infrastructure

In UniFrame, the Internet Component Broker (ICB) and Headhunters [Sira02] are proposed as two facilities in an effort to seamlessly integrate heterogeneous components. ICB provides translation capacity in terms of adapter technology for achieving interoperability, while Headhunters actively detect the presence of new components in the search space, register their functionality and attempt match-making between client components (service requesters) and server components (service providers). By generating such component specifications in XML, a component can be exposed for external querying, e.g., using XQuery[2]. Also, a pre-built meta-model, from which the domain-specific model is created, represents the domain ontology [Grub93] and provides the leverage for the ICB and Headhunter.

The aforementioned three concerns necessitate a proper methodology of creating a meta-model to modeling the following categories:

Table 1. Component Description in UMM

Computational Attributes	Inherent Attributes	ID
	Functional Attributes	Description
		Algorithm
		Complexity
		Syntactic Contract
		Technology
		…
Cooperation Attributes	Precondition	
	Postcondition	
Auxiliary Attributes	Security	
	Mobility	
	….	
QoS Metrics	Availability	
	End-to-End delay	
	……	

[1] http://www.w3.org/2002/ws/

[2] http://www.w3.org/XML/Query

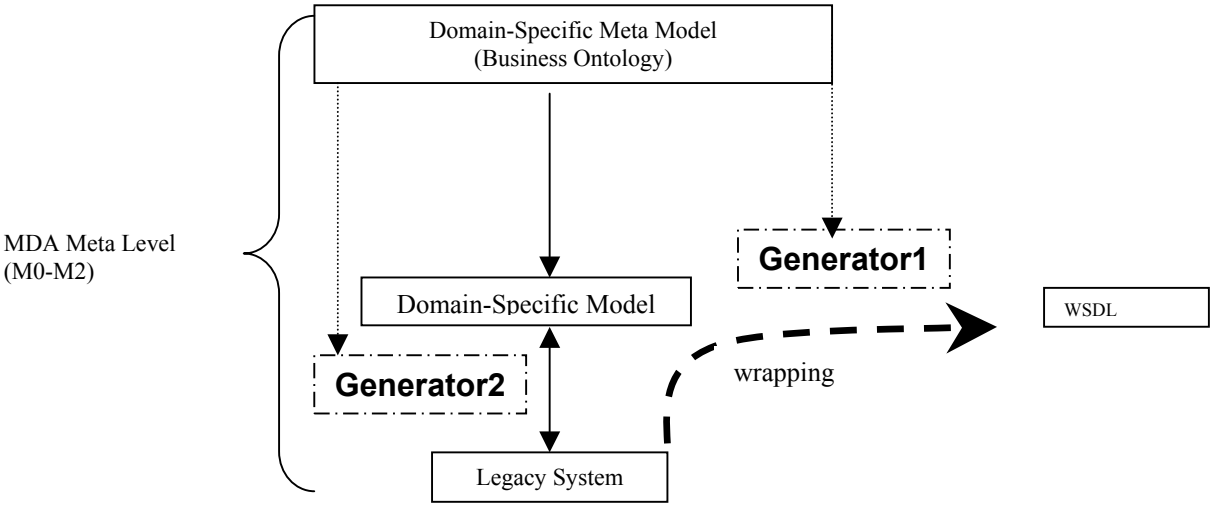


Figure 2. Overview of Approach

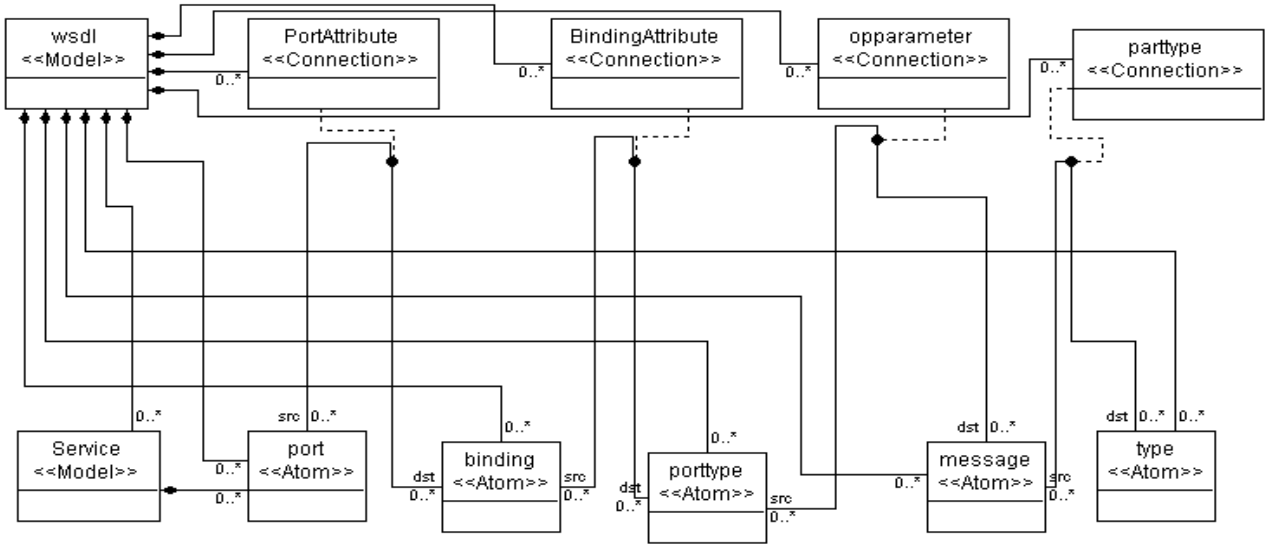


Figure 3. Meta-Model of WSDL

Obviously, a pure textual specification of UMM, while still a viable choice, will be error prone and hard to be processed and reused. The widely used Rational Rose [Quat00] toolkits, however, can only be used for non-executable modeling, in the sense that you have no control over generation of complete applications, which is not adequate enough for modeling UMM. This problem will be addressed using tool support introduced in the next section.

3. Modeling as the Front End of Web Services

3.1 Generic Modeling Environment (GME)

Model Integrated Computing (MIC) employs meta-modeling to define the domain modeling language and model integrity constraints. It uses these meta-models to automatically compose a domain-specific design environment and generate input to some analysis tools such as Matlab Simulink/Stateflow [Neem02]. MIC includes the Generic Modeling Environment (GME) for creation of domain-specific models, a Model Database for model storage, and a Model Interpretation technology for building model interpreters. In GME, the meta-models use Unified Modeling Language (UML) class diagrams to model the system information. Figure 3 gives the WSDL meta-model using GME. Also MCL (MGA[1] Constraint Language) [GME00], which is a subset of UML OCL[2] with some MGA specific extension, is used to enforce some

[1] MultiGraph Architecture [Szti95]

[2] http://www-3.ibm.com/software/ad/library/standards/ocl.html

semantic rules in MGA modeling paradigms. This adds some formalism to the modeling, which can be used to enrich the semantic expressiveness of WSDL, as is explained later in section 5

WSDL is not convenient to be manually coded. Many tools such as AXIS[1], and the Microsoft .Net framework provide the function of generating WSDL from implementation code (such as Java and C#) and vice versa. Such tools leverage compiler technology to generate WSDL from some other programming languages. In contrast, by generating WSDL from a high-level language-independent model, we can avoid the need for language-specific compilers. This permits easier maneuvering of the generated WSDL at a higher level. Also, by standardizing the meta-model and the associated generator, the domain ontology will be uniformly embodied in generated WSDL. This will facilitate program-to-program interoperation bearing the intelligence of software agents, such as autonomy and knowledge [Gris01].

3.2 Enriching and Modeling WS Semantics

Current WS standards mainly embody the semantics of processes at the collaborating syntactic interface level. WSDL only exposes distributed object services, while such process behavior aspects as ordering, and dependency are not well specified in the existing WSDL standard. Figure 4 gives the meta-model of a Finite State Machine (FSM), which can be used to model the dynamic behavior of WS, in particular, the sequence of states that the WS behavior goes through in its lifetime. We will illustrate this point in detail in a later example.

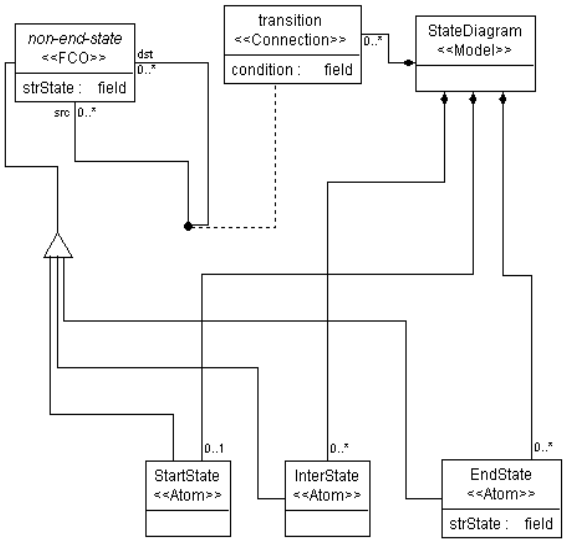


Figure 4. Finite State Machine (FSM) Meta-model

[1] http://ws.apache.org/axis/

4. Web Services Generator

A key aspect of MDA is the generator technology. By generating implementation code from a high-level specification language, software systems can be produced with high efficiency while the scale of software reuse will be reduced at the specification level. GME provides the Builder Object Network (BON) framework [GME00] for building interpreters by instantiating each object in the model tree with a C++ object. The objects in the model tree can be traversed by calling methods within the BON API. In order to precisely generate target code from the models using a generator, a special *atom* can be added in the GME environment denoting specific meaning so as to enrich the semantics of modeling. e.g., in feature modeling [Czar00], there are mandatory features, optional features and alternative features for some concept. We can add a *Require* atom, an *Or* atom, an *XOR* atom to denote the three relationships between other atoms. Figure 5 illustrates the strategy. In this way, the designated semantics can be captured when traversing the model tree. This strategy can also be applied to model UML relationships such as *Dependency*, *Generalization*, *and Association*. In this way, the built-in class diagram facilities of GME itself can be extended.

5. Putting it Together

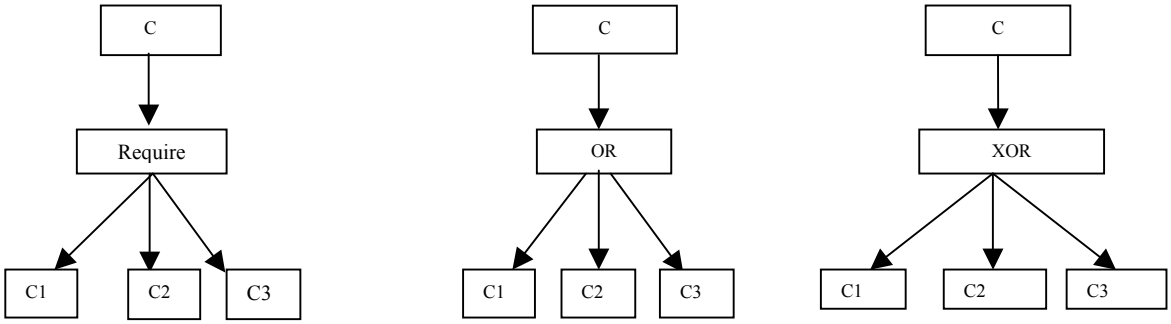
This section will use GME to create a meta-model embracing both UMM and WS, and an interpreter is built based on this meta-model for generating WSDL in an effort to facilitate component service synthesis in UniFrame.

5.1 Creating Banking Domain Meta-Model

Below is a simple banking domain specification:

```
A bank provides the service for users to
set up accounts.  Account information
includes personal data including Name,
SSN, phone number, address, and account
data including Account Number, PIN,
Transaction Record, Balance.  There are
two types of accounts: checking account
and savings account.

For the bank side, it provides such
services as: Account Validation (to
ensure legal access of account), Account
Verification (to double check the account
after each transaction, including
transaction history, transaction
description, etc), Account Query (balance
checking), Deposit, Withdraw, and
Transfer. There is order restriction for
those operations. Both Transfer and
Withdraw have to be preceded by a Query
operation. The Account Verification comes
after each of the other operations.
```



(1) Concept C requires all the three features: C1, C2, C3

(2) Concept C contains some of the three features: C1, C2, C3

(3) Concept C contains exactly one of the three features: C1, C2, C3

Figure 5. Representing Semantics of Feature Modeling with Atom-to-Atom Connection

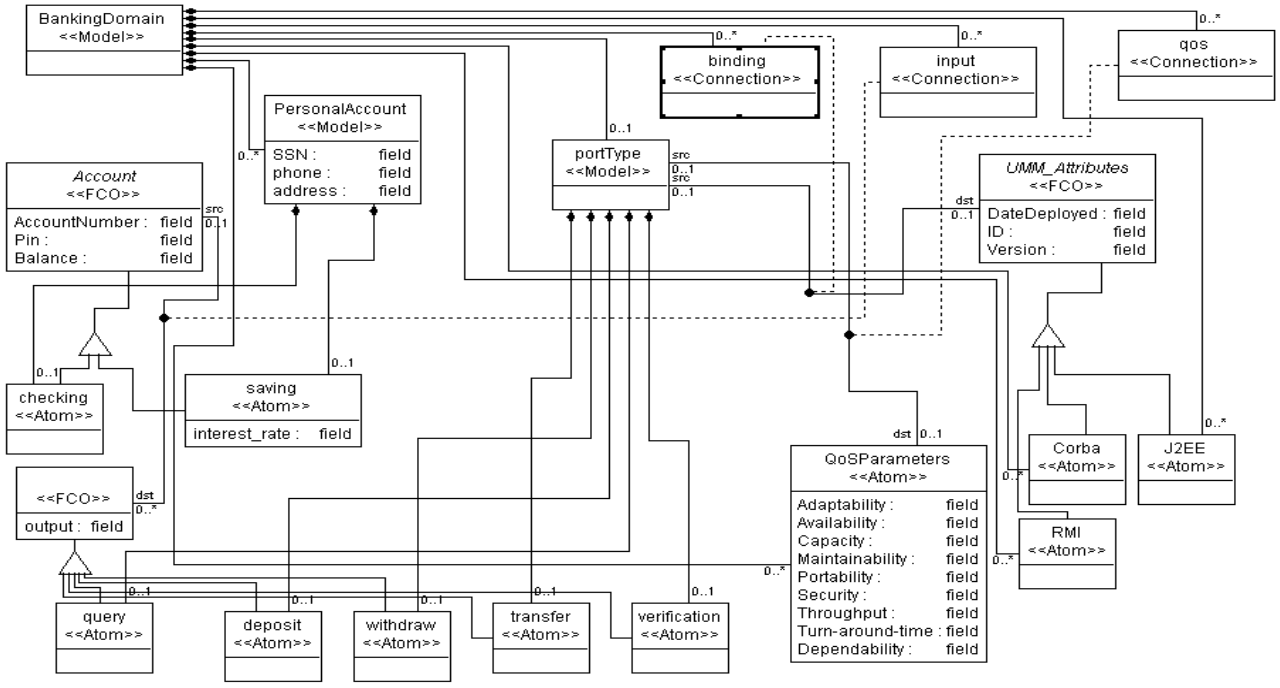


Figure 6. Meta-model of Banking Domain

Deposit and withdraw can only be applied to checking account (this is not the generic case, though). The aforementioned services are optional so long as the above rules are observed.

The banking service may leverage such technology as RMI, J2EE, and CORBA. Also it will enforce some QoS concerns such as Availability, Dependability, Capacity, etc. (For more QoS parameters see [Raje02]).

Directly expressing the above specification in WSDL will tend to blur the 4+1 view[1] of the software architecture. Thus it is hard to represent the intended requirements precisely and the constraints can not be warranted. Model-based WSDL generation will be able to solve the ambiguity problem by clearly modeling the specification in a graphical fashion to capture all the involved relationships. The meta-model in Figure 6 represents the banking domain knowledge. It's derived from WSDL elements and banking domain knowledge. *portType* in WSDL denotes the WS abstract interface definition. It is represented as a *model* in Figure 6, which contains the following banking-domain specific operations: *query*, *deposit*, *withdraw*, *transfer*, *verification*. *binding* in WSDL denotes how the elements in an abstract interface (*portType*) are converted into a concrete representation in a particular combination of data formats and protocols (here, platform specific implementation in

[1] which includes functional requirements, software module organization, run-time implementation structure of the system, etc. For details see [Kruc95]

CORBA, J2EE, RMI, etc). Consequently, *binding* is represented as a *connection* between *portType* and *UMM_Attibutes*, which is the parent of the CORBA, J2EE and RMI *atom*s.

The left part of Figure 6 (*PersonalAccount*, *Account*, *checking*, *saving*) is basically about a simplified version of the feature modeling [Czar00] of the banking domain, which is treated as input (represented as *connection* here) into operations of *portType*. Also QoS parameters, by being associated with *portType*, will be embedded into the generated WSDL as extended attributes. WSDL itself is XML based, so a query expressed in XQuery can make use of extended WSDL attributes to refine the query in selecting targeted WSDL. Here, the listed QoS parameters are treated as of static type. For dynamic parameters, we can apply aspect weaving [Kicz97] technology in the code generation phase for performing dynamic measurements.

The specified constraints over *withdraw* and *deposit* operations can be enforced in GME using the following MCL (refer back to section 3.1) expression:

```
connectedFCOs("src")->forAll(
        c|c. kindName() ="checking")
```

Those constraints apply to both the *withdraw* atom and the *deposit* atom in Figure 6, which means those First Class Objects (FCO: referring to both entities and relations in GME) that are connected with *withdraw*/*deposit* atoms are all of kind "checking"; i.e., those services can only be applied to *checking account*.

But, when it comes to the handling of order constraints as specified in the banking domain example, obviously MCL is not adequate enough to capture such dynamic behaviors. Such modeling techniques as using the Finite State Machine will provide modeling capacity for advanced behavior, which is detailed in the next section.

5.2 A Banking Model and WS-based Integration

Figure 7 is an example of the banking model. For this model, "My Account" is the name for the "PersonalAccount" model. It has two kinds of account: both checking (c) and savings (s). "Service Offering" represents the "portType". It offers 4 types of service (without transfer in this case): d: *deposit*, q: *query*, w: *withdraw*, v: *verification*. From the connections between the ports we can see for this banking model, the *query* can only be applied to the savings account, while *verification* can be carried out over both types of account. *Withdraw* and *deposit* only applies to checking account. Otherwise the modeling environment will give warnings when modeling, which is consistent with the MCL specification. Also, notice for this banking model, RMI technology is adopted and some QoS parameters are specified here, as shown in the lower-right corner attribute list. The attribute list associated with RMI will also be shown in the corner if the RMI atom is under focus.

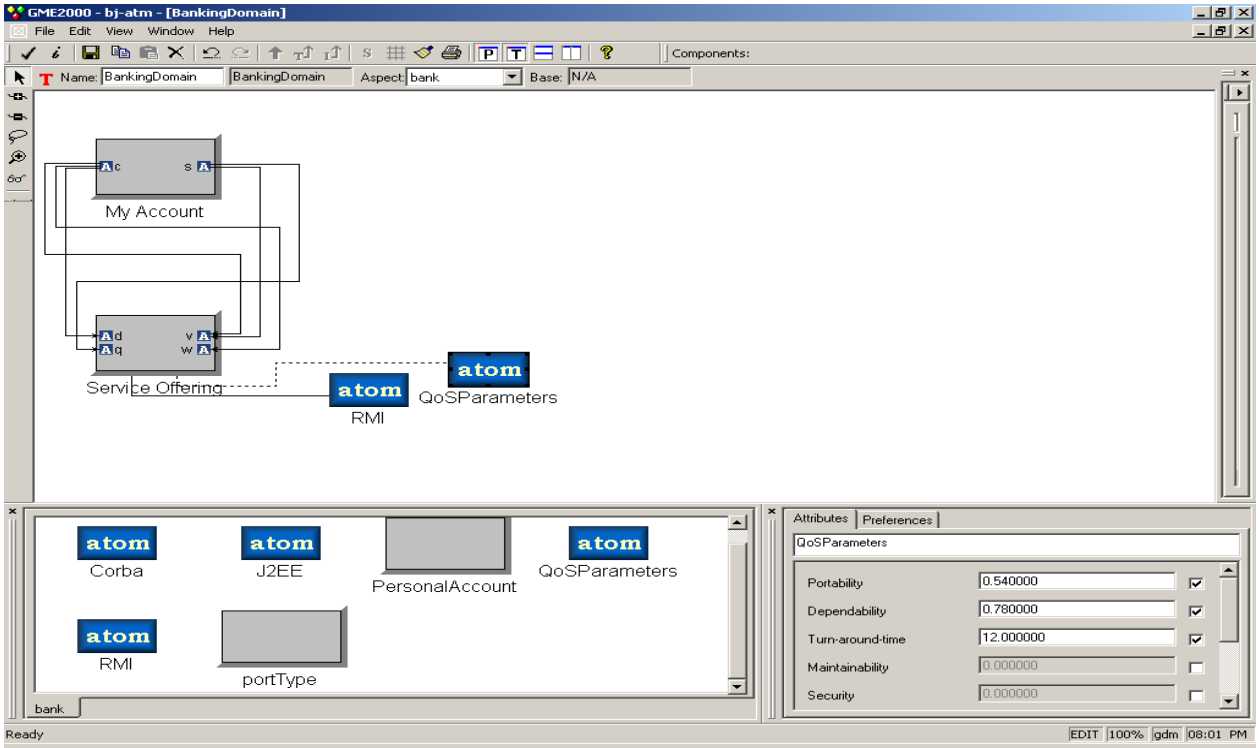


Figure 7. "My Account": a Banking Model

From the model in Figure 7 the interpreter will generate two sets of codes: the WSDL code for the banking service embedded with QoS parameter extension, and the WS wrapping code for the underlying RMI implementation. Because the generated WSDL is quite lengthy, we will just show some model-specific contents as shown in the following paragraph. Notice the bold-font part of the following WSDL represents the QoS extension of WSDL, which may be used for WS filtering if QoS requirements are submitted in the query expression.

```
<definition name="my bank">

<types>
  <xsd:schema
   targetedNamespace="http://localhost/bank"
   xmlns:xsd="http://www.w3
        .org/2001/XMLSchema">
  <xsd:complexType name="Account">
      <xsd:sequence>
         <xsd:element  name="AccountNumber"
           type="xsd:string"/>
          <xsd:element name="Pin"
              type="xsd:string"/>
          <xsd:element name="Balance"
              type="xsd:decimal"/>
      </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="checking">
      <xsd:complexContent>
          <xsd:extension  base="Account">
      </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="savings">
   <xsd:complexContent>
           <xsd:extension base="Account">
    <xsd:attribute name="interest_rate"
     type="xsd:decimal"/>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:schema>
</types>

<message name="checking">
 <part name="p1" type="checking"/>
</message>
<message name="savings">
 <part name="p1" type="savings"/>
</message>
<message name="checking_savings">
 <part name="p1" type="checking"/>
 <part name="p2" type="savings"/>
</message>

<portType name="bankPortType">
    <operation name="withdraw">
        <input message="checking"/>
       <output message=""/>
    </operation>
   <operation name="deposit">
       <input message="checking"/>
       <output message=""/>
   </operation>
  <operation name="verification">
   <input message="checking_savings"/>
   <output message=""/>
  </operation>
  <operation name="query">
     <input message="savings"/>
      <output message=""/>
  </operation>
</portType>

<binding>
.........
</binding>

<service name="My Bank" Portability="0.544400"
   Dependability="0.780000" Turn-around-
   time="12.000000"/>
  <port>
 .....
  </port>
</service>

</definition>
```

Now we turn to the handling of the order restriction requirement in the banking domain specification. We will use the FSM meta-model (Figure 4) to build the banking service state model as shown in Figure 8 and the associated interpreter. Because every service corresponds to the child node (atom) of *portType model* in Figure 6, we can use BON API (refer back to Section 4) to traverse those child *atoms* of *portType* in the banking model one by one while retrieving the connection information of each *atom*. The generated WSDL extension describing the state transition process is as follows:

```
<state>
   <state name= "Login" >
   <state name="Validation" >
   <state name="Query" >
   <state name="Deposit" >
   <state name="Transfer" >
   <state name="Withdraw" >
   <state name="Verification" >
</state>
<transition>
  <transition src="StartState"
   dst="Login" condition="">
  <transition src="Login" dst="Login"
   condition="">
  <transition src="Login"
   dst="Validation" condition="">
  <transition src="Validation"
    dst="Deposit" condition="">
  <transition src="Validation"
   dst="Query" condition="">
 <transition src="Deposit" dst="Deposit"
   condition="">
  <transition src="Deposit"
    dst="Verification" condition="">
  <transition src="Query" dst="Transfer"
     condition="">
  <transition src="Query" dst="Query"
    condition="">
  <transition src="Query" dst="Withdraw"
    condition="">
  <transition src="Query"
    dst="Verification" condition="">
  <transition src="Transfer"
    dst="Transfer" condition="">
  <transition src="Transfer"
    dst="Verification" condition="">
  <transition src="Verification"
    dst="StartState" condition="">
  <transition src="Verification"
     dst="Verification" condition="">
  <transition src="Verification"
    dst="EndState" condition="">
  <transition src="WithDraw"
   dst="WithDraw"  condition="">
  <transition src="WithDraw"
    dst="Verification" condition="">
</transition>
```

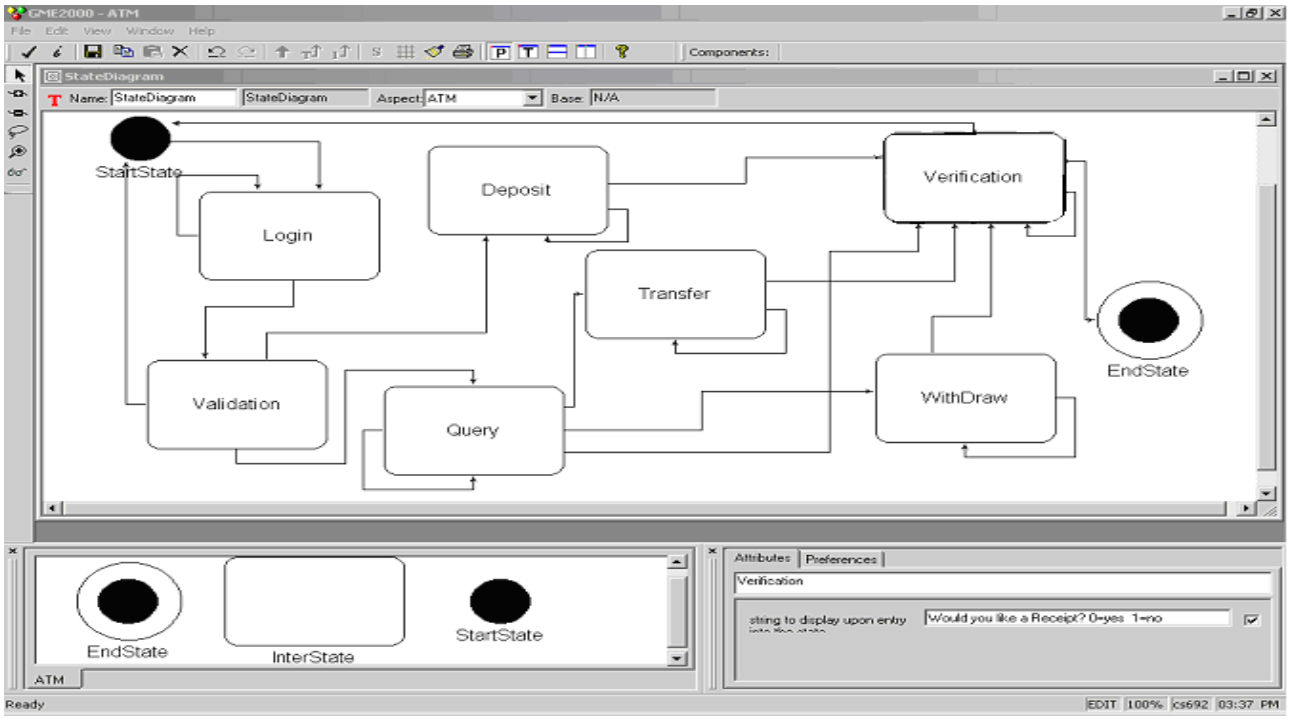


Figure 8. Banking behavior model based on FSM meta-model

Note in the generated state transition code, the "condition" attributes are supposed to be customized in the specific banking behavior model before code generation, which for the sake of brevity are left blank here. The state transition specification generated here may be used in guiding the WS consumption and composition.

6. Conclusions and Future Research

This paper applies the model driven approach to WS technology. By modeling service behavior at a higher level, the system semantics can captured at a finer grain. Meanwhile, different artifacts can be derived from models using a generator, which will not only refine the service presentation, but also facilitate system integration. In particular, this approach is applied in the context of the UniFrame project for system integration. So far, we have implemented a prototype with the function of WSDL generation from a specific component model and FSM modeling for component services.

Because the meta-model is the starting point and cornerstone of system integration, we will need to refine the meta-model leveraging domain knowledge until it can be standardized. To enhance the semantics expressing capability of WS, future research will involve not only state machine modeling, but also the modeling of other behavior concerns, such as interaction, activity, process/thread and temporal relationship. Also, technology and QoS modeling in the above banking example are still quite primitive, both of which need further exploration for the ultimate model-based glue/wrapper code generation between WS and other component models.

Acknowledgements. This research is supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

REFERENCES

[Brya02] Bryant, B. R., Auguston, M., Raje , R. R., Burt, C. C., Olson , A. M., 2002, "Formal Specification of Generative Component Assembly Using Two-Level Grammar," *Proc. SEKE, 14th Int. Conf. Software Engineering and Knowledge Engineering,* pp. 209-212.

[Czar00] Czarnecki, K., Eisenecker, U.W., 2000, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley.

[GME00] "GME 2000 User's Manual, Version 2.0," 2001, ISIS, Vanderbilt University.

[Grah02] Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura,Y., Neyama, R., 2002, *Building Web Services with Java*, SAMS.

[Gris01] Griss, M., 2001, "Software Agents as Next Generation Software Components", *Component-Based Software Engineering*, ed. Heineman, G. T., Councill, W. T., Addison-Wesley, pp. 641-657.

[Grub93] Gruber, T. R., 1993, "A translation approach to portable ontology specifications," *Knowledge Acquisition,* Vol. 5, No. 2, pp.

199-220.

[Kicz97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., and Irwin, J., 1997, "Aspect-Oriented Programming," *Proc. ECOOP, European Conference on Object-Oriented Programming*, Springer-Verlag LNCS Vol. 1241, pp. 220-242.

[Kruc95] Kruchten, P.B., 1995, "The 4+1 Views Model of Architecture", *IEEE Software*, Vol. 12, No. 6, pp. 42-50.

[Léde01] Lédeczi, Á., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J. and. Karsai, G., 2001, "Composing Domain-Specific Design Environments," *IEEE Computer*, Vol. 34, No. 11, pp. 44-51.

[Medv97]Medvidovic, N., Taylor, R.N., 1997, "A Framework for Classifying and Comparing Software Architecture Description Languages, " *Proc. ESEC/FSE '9, European Software Engineering Conf./9th Conf. Foundations of Software Engineering,* Springer-Verlag LNCS Vol. 1301.

[Neem02] Neema, S., Bapty, T., Gray, J., Gokhale, A., 2002, "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems," *Proc. GPCE, First ACM SIGPLAN/SIGSOFT Conf. Generative Programming and Component Engineering,* Springer-Verlag LNCS Vol. 2487, pp. 236-251.

[Quat00] Quatrani, T., 2000, *Visual Modeling with Rational Rose 2000 and UML*, Addison Wesley.

[Raje00] Raje, R., 2000, "UMM: Unified Meta-object Model for Open Distributed Systems," *Proc. ICA3PP, 4th IEEE Int. Conf. Algorithms and Architecture for Parallel Processing*, pp. 454-465.

[Raje01] Raje, R., Bryant, B., Auguston, M., Olson, A., Burt, C., 2001, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components," *Proc. Monterey Workshop Engineering Automation for Software Intensive System Integration*, pp. 109-119.

[Raje02] Raje, R. R., Auguston, M., Bryant, B. R., Olson, A. M., Burt, C. C., 2002, "A Quality of Service-Based Framework for Creating Distributed Heterogeneous Software Components," *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 2, pp. 1009-1034.

[Sira02] Siram, N. N., Raje, R. R., Olson, A. M., Bryant, B. R., Burt, C. C., Auguston, M., 2002, "An Architecture for the UniFrame Resource Discovery Service," *Proc. SEM, 3rd Int. Workshop Software Engineering and Middleware,* Springer-Verlag LNCS Vol. 2596.

[Szti95] Sztipanovits, J., Karsai, G., Biegl, C., Bapty, T., Lédeczi, Á., Misra, A., 1995, "MULTIGRAPH: An Architecture for Model-Integrated Computing," *Proc. IEEE ICECCS, International Conference on Engineering of Complex Computer Systems*, pp. 361-368.