Managing Business Logic with Symbolic Computation

Ruslan Schevchenko, Anatoliy Doroshenko
Institute of Software Systems
National Academy of Sciences of Ukraine
Glushkov prosp. 40, 03187 Kiev, Ukraine
Email: rssh@gradsoft.com.ua, dor@isofts.kiev.ua

Abstract: A new approach of building enterprise distributed applications for layer of business rules as standard part of architecture is proposed in this paper. The approach uses symbolic transformation system based on rewriting techniques and embedded in Java language and provides analysis and engineering distributed applications at source code level. The benefit of the system is extreme flexibility of the approach and light-weightedness of formal methods and their separate application to program features that does not need to know full semantic model of application programs. The system is implemented and its usage is illustrated with an application example of engineering CORBA middleware application.

1. Introduction

Contemporary distributed business applications are characterised by complex behaviour and dynamical evolving in changing environments. To avoid possible inconsistencies as well as to make improvements of non-functional properties of application some instrumental facilities are needed in support of program manipulation.

Typical structure of business application consists of following main parts:

- *persistent layer* that is usually managed by relational database;
- *business logic* which consists of state represented by language-specific object model and behaviour expressed in 3GL language;
- *presentation layer* which is usually build with a help of RAD tools or scripting techniques.

All of this is bundled in one package with help of some standard infrastructure such as application server or CCM deployment container. During a process of application development we have high-level modelling techniques for object architecture (UML, OOD) which allow developers to describe application domain quickly and cleanly, and we have RAD techniques for building user interfaces. But behaviour of application usually coded by hands. Usually we have no formal models and RAD techniques in one place and at the same time. So if a question arises 'what our application must do?' we often fail to answer. Existing methods are not sufficient to be applicable in practice due to following reasons. As for UML there are no constructive methods of mapping of UML use-cases and UML state and sequence diagrams are too low level and describe physics but not logic. As for other facilities it should be noted that IDEF [IICEE95],[IDEF95] are non-constructive and WSDL [WSDL01] or BPML [Aa02] have cost of writing and maintenance of specification that is comparable to cost of coding this behaviour in some programming language.

So one can conclude that describing behaviour is a bottleneck of today technique of application development. In this paper we propose a new approach to building enterprise distributed applications that use symbolic computation for layer of business rules as standard part of architecture. For this purpose we have found to use convenient formal program transformations by means of symbolic computation facilities.

Source code formal transformations are widely recognised as high-level and powerful facility that gives advantages of strong foundation of source code analysis and in many cases can provide full automation of solving problems in program manipulation and software maintenance. On one extreme of a spectrum of formal transformation methodologies there is full- fledged systems based on wide spectrum language. Known example of such kind of systems is FermaT transformation system [Wa95] – an industrial-strength transformation system designed for forward and reverse engineering and program comprehension. However despite knowledgeability of this approach complete utilisation of its advantages is very complicated, expensive and needs extensive support at all stages/levels of software evolution. On the other extreme in recent years there are of ever growing interest to "light-weighted" approaches to formal methods for program analysis aimed to reduce costs and improve efficiency of software maintenance. They try to focus attention to particular features of programs where formal (symbolic) computation can be both a good base for modelling/specification purposes and, on another hand, an efficient tool for computation of accurate program information avoiding full formalisation of program semantics (see for example [Li00]). The latter approach is particularly valuable in context of dominating trend of component-oriented software development that assumes building new software system from pre-existing components with some glue between components and new functionality. To reuse components effectively we need some facilities to capture interface specifications of components and methods of interface compatibility validation. Formalisms like UML [UM00] widely used in design and documentation appear too informal and heavy for these purposes. Recently a symbolic computation system (named TermWare) based on rewriting techniques has been proposed to provide efficient and cost effective reengineering program source code by means of formal transformations [SD03],[RS02]. Our primary goal for the system was to have impact on software engineering efficiency aspects inspired by our previous experience in development of industrial-strength projects of CORBA based software [SD02],[SD01],[SH01]. It appeared however that Termware can be used for managing business logic and improvements in interface mechanisms described in languages like CORBA's IDL [CO99]. Usually, IDL can provide a kind of general guarantee of software component interoperability for customers. But to ensure user confidence in software services in particular context and on particular network platform some additional efforts are needed. We posed the task to make this process automatically and at low cost. The system usage is demonstrated in this paper by a case study with representative example in business logic engineering of distributed program code.

2. TermWare: embedding symbolic logic into application

TermWare system [RS02] is a term-rewriting extensible framework consisted of two main parts: 1) Java library which containing basic data structures and algorithms for rewriting techniques like terms, term rewriting rules , unification, rewriting strategies and others; and 2) Java framework, containing facilities for adding parsers, programming Languages and rewriting rules with actions which can be embedded into Java application and can be extended via Java framework. We refer interested readers to [RS02] for formal semantics of TermWare and present here only brief description of its main concepts and features. In the world of rewriting systems TermWare is positioned between ATerm+ASDF [Br00] and FermaT [Wa95] from one side, and Maude [Wi93] and APS [LKK93] from the other side. Also it is closed to other fundamental approaches in contemporary directions software design such as strategic programming [Vi01] and design patterns [Ga95]. In contrast to ATerm and FermaT term systems, terms in TermWare are not 'just trees', but natural logic terms and in addition to tree traversing we have embedded unification, propositional variables and so on. The main difference of TermWare from conventional term rewriting systems like Maude and APS consists in that term rewriting system is not 'closed' formal system in the sense that it does not intend to provide complete programming environment; and it is packaged not as interpreter or translator, but as a library for embedding. TermSystem is the main concept in TermWare like class in object-oriented programming. Unlike traditional set of rewriting rules it extends functionality for interaction with external world, reprsented as deductive term database.

2.1 Embedding application semantics into logic framework

Usually we think about application in terms of object model, and when we think about business rules we think in terms of logic. The question is how logic can be applied to behaviour of application. To answer this question developers usually are governed by following observations:

1. Persistence and entity beans or layers can be mapped to logic framework such as knowledge database.
2. Presentation layer can be mapped to oracles.

So, we have not just logic but logic with environment. From systems theory point of view, application behaviour can be viewed as function: $X \times E \times S \rightarrow Y \times S \times E'$ where X, Y are input and output reactions respectively, S is set of states and E is environment. And the process of defining application domain logic can be presented as sequence of steps: 1) mapping inputs to set of possible input terms; 2) mapping actions (outputs) to set of possible actions; and 3) mapping domain entity layer to knowledge database.

High-level presentation of TermWare system can be seen as a quadruple $< S,E, \phi_s, \phi_e>$ where $S=< S_t, S_r>$ is pair of set of terms S_t and active set of rewriting rules S_r, E is environment representing in the system; $\phi_s : S \times X \times E \rightarrow S \times Y$ is the system transformation function. ϕ_e: $E \times Y \rightarrow E$ is the environment reaction function. System transformation function is defined by set of rules being applied according to some strategy. System transformation function is defined by set of rules, which are applied

according to some strategy. Rule is a quadruple of the form $(x, e_{in}) \rightarrow (y, e_{out})$, where x, y are input and output terms, e_{in} is environment state request and e_{out} is environment influence operation.

2.2 Embedding logic framework into application infrastructure

Now look at technical side of the task: how rules can be evaluated inside application. TermWare is distributed not as standalone executable but as Java library which provide functionality to build and evaluate so-called 'Term Systems'. Term System is consists of following parts:

- string that is the name of Term System;
- sequence of conditional rewriting rules with actions;
- a strategy of applying such set of rules to terms; (a strategy is just an algorithm for controlling sequence of reductions, coded in Java);
- facts of database, that play the role of oracles in TermWare logic; conditions in TermWare can request information from database and actions can change information in database; on Java level facts are instances of interface IFact.

TermSystems are organised into hierarchical namespace. We can reuse their definitions in object-oriented manner and create new ones with special operators such as superposition or Cartesian product. Java application developer see term systems as instances of class `ItermSystem` with following signature:

```
public class ITermSystem
{
public ITermSystem(ITermRewritingStrategy strategy,

                   IFacts facts,
                   IEnv   env);

  ...

  public boolean checkFact(ITerm t) throws
TermWareException;

  public void setFact(ITerm t) throws TermWareException;

  public void addRule(ITerm t) throws TermWareException;

  public ITerm reduce(ITerm x) throws TermWareException;

  ......

};
```

`ITerm` is a Java incarnation of `term`. Application developer can create instances of

term from strings or input streams using TermWare parser, embedded in the library.

Knowledge database is represented as interface IFacts:

```
public interface IFacts
{

 public String   getDomainName();

 public boolean  check(ITerm t) throws TermWareException;

 public ITerm    ask(ITerm t) throws TermWareException;

 public void     set(ITerm t) throws TermWareException;

 public void     remove(ITerm t) throws TermWareException;

}
```

Typical work of IFacts is to map term operations to application specific database and user-interaction requests. Strategy is an algorithm for applying of rewriting rules. TermWare provide set of predefined strategies which can be used.

IEnv encapsulate application infrastructure which must be accessible during processing, such as logging or input/output.

3. An example of application

So, what is the approach of using TermWare in application software? The main steps are following:

1. Build logic framework which represent the properties of domain objects as Termware terms, i. e. build some ontology.
2. Define mapping of TermWare logic queries to relational database queries.
3. Define mapping of actions to call of application-depended workflow engine.
4. Describe current set of business processes and constraints as TermWare rewriting rules.

Let us describe business process of software package maintenance in development organisation:

```
System(BugFixing,DevelopmentProcess,
 ruleset(

 received($bug_id) -> check_confirmation($bug_id)
                                  //
human_task(check_bug($bug_id)),

 check_confirmation($bug_id) [|confirmed($tester,$bug_id)|]
```

```
                          -> known($bug_id)
                                //
human_task(fix_bug($bug_id),

write_regression($bug_id)
                                                  ),

 check_confirmation($bug_id)
[|not_confirmed($tester,$bug_id)|]
                          -> true //
send_not_confirmed($bug_id,$tester),


 known($bug_id) [|
                  fixed($developer1, $bug_id) &&
                  added_regression($developer2,$bug_id)
                 |]
                          -> true //
send_closed($bug_id,$developer1)
 ),
 FirstTop)
```

This example describes TermWare application in business-processes organisation system. A reader might guess that it is an example of software error message processing. Here fact base (environment) can answer following questions:

- *confirmed(x,y)* - *x* person confirmed *y* error message;
- *not_confirmed(x,y)* - *x* denied *y* message;
- *fixed(x,y)* - *x* repaired software *y* error;
- *added_regression(x,y)* - *x* added tests for *y* into regression tests set.

and can interpret following messages:

- *human_task(x)* - ask somebody to perform *x;*
- *send_not_confirmed(x)* - report about denied error *x;*
- *send_closed(x)* - report about repaired error *x.*

Notice that environment data are transmitted to the system by sentential expressions in conditions.

On the Java side mapping of unification to work with relational database can be expressed in following way:

```
class DevelopmentProcessFacts extends DefaultFacts
{

  /*
  * $x$ person confirmed $y$ error message
  */
  boolean check_confirmed(ITerm x,ITerm y) throws
```

```
TermWareException
  {
   BindSet bs= getEnv().dbPool().createBindSet();
   bs.add(y.getAsInt());
   ResultSet rs = getEnv().dbPool().evaluate(
        "select confirmator_id from bugs_confirmed where
bug_id=?"
                                                );
   if (rs.getLength()==0) return false;
   else if (x.isX()) {
     x.set(ITermParser.createInt(rs.getAsInt(0)));
     return true;
   }else {
     return x.getAsInt()==rs.getAsInt(0);
   }
  }

..........

};
```

Above `check_confirmed` is called during term processing via Java reflection API. Interaction with user is organised with a help of application-specific workflow infrastructure:

```
class DevelopmentProcessFacts extends DefaultFacts
{

 ..............

  /*
  * ask somebody to perform $x$
  */
  boolean set_human_task(ITerm x) throws TermWareException
  {
   for(int i=0; i<x.arity(); ++i) {

getEnv().workflow().put("unassigned",generateTaskURL(x.getS
ubtermAt(0)));
   }
  }

..........

};
```

Once we define set of operations, we can change business logic in any point of

application life time.

4. Conclusion

In this paper a new approach of building business logic layer is proposed with help of high-level language using symbolic computation framework. The essence of our approach consists in involving facilities to describe and evaluate logical semantics of application. This can change traditional way of application development and can have several advantages. Fist, a bottleneck of todays technology, permanent programming application logic, can be significantly reduced and in many cases even eliminated and substituted by logic component reuse. Then, decoupling behaviour and architecture parts of applications allows to change business rules during application life time. So business reorganization not longer means rewriting of software. At last process of application deployment can be interactive.

Future of investigation of this approach can be directed to logical verification of business rules, automating mapping of logic queries to relational databases and building bridges between logic rules and some graphical notation of business processes, such as IDEF or UML workflow profile. Integration with existing ontology building methods will be helpful. Also we have some intention to apply Symbolic Transformations Framework for more general Distributed Systems Analysis.

TermWare system is implemented in GradSoft Ltd, and in time of writing this paper is in progress to be adopted in a number of actual projects. More detailed description of TermWare and some applications are available from Grad-Soft web page: http://www.gradsoft.com.ua

Bibliography

[Aa02] Arkin, A.: Intalio. Business Process Modeling Language. bpml.org. November 13, 2002

[Br00] Brand, M. G. J. van den; de Jong, H.; Klint, P.; Olivier, P.: Efficient annotated terms. Software, Practice & Experience, 2000, 30(3); pp. 259-291.

[CO99] The Common Object Request Broker: Architecture & Specification, OMG,1999,formal/99-10-07.

[Ga95] Gamma, E.; Helm, R.; Jonhson, R.; Vlissides J.: Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley, New York, 1995.

[IDEF95] Mayer, R.J.; Painter, M.K.; Menzel, C.P., Perakath, B.; deWitte, P.S.; Blinn, T. IDEF3 Process Description Capture Method Report. Knowledge Based Systems, Inc.,l 1995.

[IICE95] Mayer, R.J.; Crump, J.W.; Fernandes, R.; Keen, A.; Painter, M.K.: Information integration for concurrent engineering (IICE) compendium of methods report. Knowledge Based Systems, Inc., 1995.

[LKK93] Letichevsky, A.A.; Kapitonova, J.V.; Konozenko, S.V.: Computations in APS. Theoretical Computer Science, 119, 1993; pp. 148-171.

[Li00] Liang, D.; Harrold, M.J.: Light-Wight Context Recovery for Efficient and Accurate Program Analysis. In Proc. 22-nd Int. Conf. Software Engineering, (June 4-11, 2000, Limerick, Ireland), ACM Press, New York, 2000; pp. 366-

406.

[SH02] Shevchenko, R.; Doroshenko, A.: A time cost model for distributed objects parallel computation. Future Generation Computer Systems, 18, 2002; pp. 807-812.

[RS02] Shevchenko, R.: TermWare: Semantics Description, GradSoft Ltd, Kiev, Ukraine, GradSoft-TermWare-e-Sm-7.10.2002.1, 2002, www.gradsoft.com.ua

[SD02] Shevchenko, R.; Doroshenko, A.: Evolution of CORBA Framework: An Experience Study. In Case studies of CSMR 2002, Workshop Proc. 6-th European Conf. on Software Maintenance and Reengineering. Budapest, Hungary, 2002; pp. 3-9.

[SD01] Shevchenko, R.; Doroshenko, A.: A Method of Mediators for Building Web Interfaces of CORBA Distributed Enterprise Applications. In (M. Godlevsky, H. Mayr, eds.): Proc. Int. Conf. ISTA-2001 on Information Systems Technology and its Applications. Lecture Notes in Informatics, vol. 4, Gesellschaft fuer Informatik, 2001; pp. 53-63.

[SH01] Shevchenko, R., Doroshenko, A.: Techniques to Increasing Performance of CORBA Parallel Distributed Applications, in PACT-2001, Proc. 6-th Int. Conf. on Parallel Computing Technologies. Lect. Notes Comput. Sci., vol. 2127, 2001; pp. 319-328.

[UM00] Unified Modeling Language (UML) v 1.4, Object Management Group, 2001,formal/2001-09-67.

[Vi01] Visser, E.: Stratego: A language for program transformation based on rewriting strategies. System description of Stratego 0.5. In (A. Middeldorp, ed.): Rewriting Techniques and Applications (RTA'01). Lect. Notes Comput. Sci., vol. 2051, 2001; pp. 357-361.

[Wa95] Ward, M., Bennett, K.H.: Formal Methods to Aid the Evolution of Software. Int. Journal of Software Engineering and Knowledge Engineering, 1995, vol. 5(1); pp 25-47.

[Wi93] Winkler, T.: Programming in OBJ and Maude, in (P. Lauer, ed.): Functional Programming, Concurrency, Simulation and Automated Reasoning. Lect Notes Compuit. Sci, vol. 693, 1993; pp. 229-277.

[WSDL01] Christener, E.; Curbera, F.; Meredith, G.; Weerawarana, S.: Web Service Definition Language. W3C Technical Report, 2001.