International Journal of
Science and Engineering Investigations vol. 1, issue 1, February 2012

Reducing Development Time Using Automated Data Transfer Object

Amir Hooshangi
(amirhoshangi@gmail.com)

Abstract-Software development time is one of the most important metrics in software industry which involves management, marketing, development teams and other vital part of the software companies. There are different factors on reducing Development time but automation and code generation are among well-grounded techniques that have been used widely. In this paper we focus on Data transfer object (DTO) in software Construction phase. While our main goal is reducing development time by Automating DTO design pattern we've studied effects of using DTO's on refactoring and design flaws. We've compared the development time by using Light weight API which automates the process in Large scale Java Enterprise Application.

Keywords-*design-patterns;development-time;code-generation*

I. INTRODUCTION

Object oriented Applications participate in major fields of computing today. These systems are widely used in distributed computing, web technologies and many other systems. Design patterns as vital building blocks of object oriented applications are noticeable. We've found that design patterns can be used to record and encourage the reuse of best practices [7]. Data transfer Object (DTO) design pattern has been used in different Java Enterprise Technologies such as EJB, Web services, file upload programs and Java messaging service [3]. One of the unique aspects of DTO in compared with other design patterns is that it heavily effects the implementation phase while it's like other design patterns in architecture point of view. While automation is not a general solution to all fields of software, studying the nature of DTO will show that using DTO in automated way is a better approach in compare with the manual way. As [8] claims it's due to the abstract level of design patterns which makes the application of design patterns a human dependent task. Beside development time there are other issues with DTO like refactoring and design flaws. We developed a light weight API for automating the process in our experimental study using Java 6.
The definitions below would help to better understanding of this paper:

- Software construction: different activities in software engineering which results the creation and maintaining software [10].
- DTO converter utility methods: DTO object carries the values of other objects and classes in its life cycle. Converting the values of main object to DTO's and vice versa for transferring DTOs could be done by using utility methods. Look at figure 3 which is sample UML class for a Student object. We've used Student class as an example for more clearance.
- POJO: Plain old java objects.

II. DTO CHARACTERISTICS

As figure 1, 2 shows DTO classes (in most cases) are simple data containers which being used to transfer data in different layers of application. A DTO class also has been used to implement Java Comparable[1] interface or having hash methods in some cases [4]. As shown in figure 4, 5 the scenario of using DTO summarizes in following steps:

- DTO classes are replica of an object (**Bean** or any **POJO**) which calls **BeanDTO** (**ObjectDTO**) and could have all or some of main **Bean** fields (based on network overheads or requirements).
- Business object is responsible for converting **BeanDTO** (**ObjectDTO**) field values to main bean (object) and vice versa and sends them to other layers of application. Look at Figure Implementing these converter methods could be time consuming task in large scale applications.
- Previous steps repeats between different layers and business object.

1- For more information look at [12]

Software developers have strong tendency to reuse designs that worked well for them [7]. Repeating these steps for every needed **Bean** (Object) causes wasting lots of time.

A. *DTO In EJB Applications*

In EJB applications business object differs if it's Entity Bean or Session Bean [5]. In this case we have update transfer object strategy (figure 6) and multiple transfer object strategy (figure 7). These strategies have effects just on the business logic of application and not on the process of using DTO.

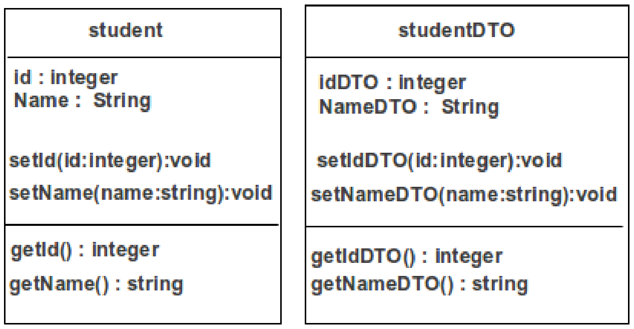

Figure 1. Student class and StudentDTO class modeled in UML.

```
public class Student {
   int id;
   String name;
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}

public class StudentDTO {

   int idDTO;
   String nameDTO;

   public int getIdDTO() {
      return idDTO;
   }
   public void setIdDTO(int id) {
      this.idDTO = id;
   }
   public String getNameDTO() {
      return nameDTO;
   }
   public void setNameDTO(String name) {
      this.nameDTO = name;
   }
}
```

Figure 2. Java codes for figure 1.

III. DTO And Issues With Refactoring, Design Flaws And Error Prone Code.

A. *Refactoring*

DTO has a tendency to change often [5]. Changing DTO classes in different construction phase affects other layers and units of application like test units, logic and etc. There are two reasons why DTO's changes a lot:

- Unclear requirements: in some situations requirement is not clear and causes to redesigning bean classes. These changes make the time consuming process.

- Naming's and coding styles: changing field and class names happens a lot in construction phase which affects the DTO classes.

```
public class StudentUtilityConverter {

public  StudentDTO  convertStudentToStudentDTO
(Student  student) {
      StudentDTO stdDTO  =  new StudentDTO();
      stdDTO.setIdDTO(student.getId());
      stdDTO.setNameDTO(student.getName());
      return  stdDTO;
   }

public  Student  convertStudentDTOToStudent(StudentDTO
studentDTO) {
      Student std = new Student();
      std.setId(studentDTO.getIdDTO());
      std.setName(studentDTO.getNameDTO());
      return  std;
   }
}
```

Figure 3. java studentDTO converter utility class.

B. *Error Prone Code*

Changing Bean classes and applying them to DTO's and converter utility classes is unstoppable cycle. This process makes error prone code. The case would be worse in large beans (with 20-30 Object properties).
Experimental study showed that it's a common mistake to setX( ) and getY( ) wrong fields in large Objects. Finding these bugs needs lots of effort because application must be checked from presentation layer to data access layers to find out where the wrong fields have been inserted.

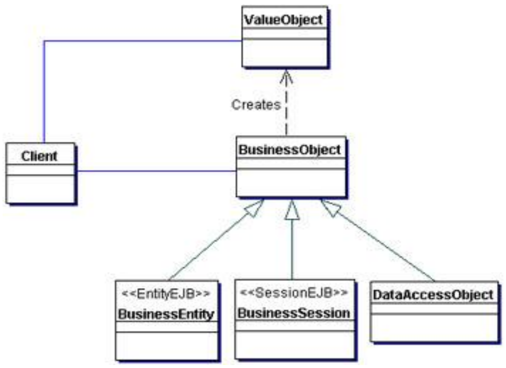

Figure 4. Transfer Object class diagram.

C. *Design Flaw*

It's a common mistake that GOF value object and DTO has been used wrongly instead of each other by developers. Also misunderstanding of DTO and assigning different design roles[2] (from other layers) causes couple and bad designed code.

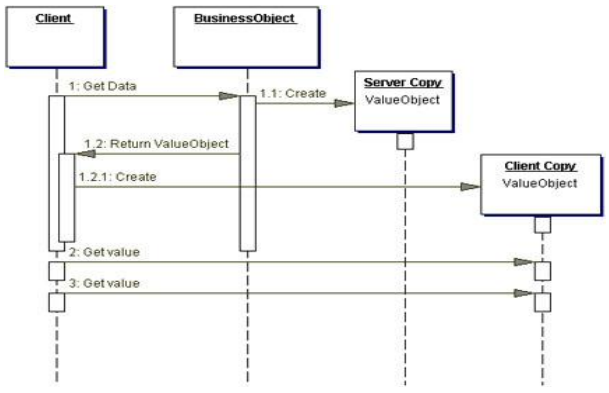

Figure 5. Transfer Object sequence diagram.

2- Design roles which assigns to the objects and classes by architecture.

IV. EXPERIMENTAL STUDY

DTO side effects on small projects are not noticeable. Studying and tracking the consumed times of our agile teams which involved Large scale enterprise applications (which used EJB and Web service technologies) showed that DTO rises as a time consumer task in different iterations of our construction phase such as requirements changes, unit testing, refactoring and etc. we used our lightweight API to re implementing the main parts of our applications and tracking the time in every iteration. As results showed, each team reported up to less 4 working days which became noticeable in long term and different iteration and phases. We are assured that there are several benefits of using DTO in automated way over manual way.

V. IMPLEMENTATION DETAILS

We developed our DTO code generator API using Java JDK 6. For generating standard and well formatted code we used Sun Code-Model [11] API. Developers are able to generate DTO classes based on Entity Beans (any POJO) by annotating required fields. By using Java reflection facilities we generate converter utility and DTO classes. For better Integration the API is also Apache Maven based.

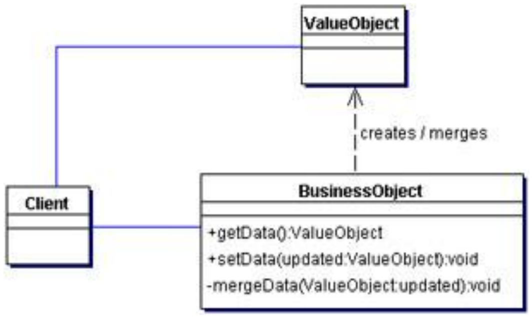
ValueObject
creates / merges
Client
BusinessObject
+getData():ValueObject
+setData(updated:ValueObject):void
-mergeData(ValueObject:updated):void

Figure 6. Update transfer object strategy.

VI. RELATED WORKS

Researchers [4] have gathered the useful information's about DTO characteristics and ways of identifying DTO's in enterprise applications by dynamic analysis. In [9] researchers have measured the effects of design roles on design on enterprise applications which describes the DTO conflicts and design flaws. IBM [6] has developed code generator for design patterns which supports C++ and Smalltalk. HTTP protocol has been used for online information's. This application also uses Perl interpreter as its engine. Indicated features are set of useful tools for analyzing and implementing design patterns in academic studies.

In [8] researchers have proposed a design patterns automation based on CBR[3] which uses UML for modeling. General approaches on design patterns are trade-off. Our work specifically targets DTO and its main goal in implementation is light weight, easy to use API for helping agility in today development teams.

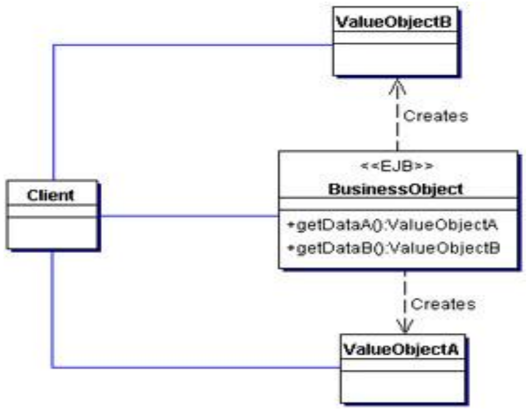

Figure 7. Multiple transfer object strategy.

VII. Conclusion And Future Works

Our study showed that using DTO in automated way is much more suitable than in manual way. In today agile teams even one working day is noticeable. Beside this, finding hidden time consumer parts of enterprise applications and removing them can cause faster available software to costumers. Future works includes improving API to implement Java Objects equals ( ) methods, hashing algorithm and supporting other features like documentations which would help DTO to be used much appropriately. Also studying other design patterns with purpose of reducing development time could be a good start to generalize our approaches on design patterns.

References

[1] Deepak Alur , Dan Malks , John Crupi, Core J2EE Patterns: Best Practices and Design Strategies, Prentice Hall PTR, Upper Saddle River, NJ, 200.

[2] http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html.

[3] Matthew A. Brown & Avery Dennison Corporation & The Pennsylvania State University CiteSeer Archives (2003). *Validation Strategy*.

[4] Alexandar Pantaleev and Atanas Rountev. Identifying Data Transfer Objects in EJB Applications. In WODA '07: Proceedings of the 5th International Workshop on Dynamic Analysis, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[5] F. Marinescu. EJB Design Patterns. John Wiley, February 2002. Alexandar Pantaleev and Atanas Rountev. Identifying Data Transfer Objects in EJB Applications. In WODA '07: Proceedings of the 5th International Workshop on Dynamic Analysis, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[6] F. J. Budinsky , M. A. Finnie , J. M. Vlissides , P. S. Yu, Automatic code generation from design patterns, IBM Systems Journal, v.35 n.2, p.151-171, 1996 [doi>10.1147/sj.352.0151] .

[7] Kent Beck , Ron Crocker , Gerard Meszaros , John Vlissides , James O. Coplien , Lutz Dominick , Frances Paulisch, Industrial experience with design patterns, Proceedings of the 18th international conference on Software engineering, p.103-114, March 25-29, 1996, Berlin, Germany .

[8] P. Gomes, F.C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferreira, and C. Bento. Using CBR for Automation of Software Design Patterns. Lecture Notes in Computer Science, pages 534--548, 2002.

[9] Cristina Marinescu. Identification of design roles for the assessment of design quality in enterprise applications. In Proceedings of International Conference on Program Comprehension (ICPC 2006), pages 169–180, Los Alamitos CA, 2006. IEEE Computer Society Press.

[10] Steve McConnell, Code complete: a practical handbook of Software construction, Microsoft Press, Redmond, WA,1993.

[11] http://codemodel.java.net/

[12] http://download.oracle.com/javase/6/docs/api/java/lang/Comparable.html.

3- Case Based Reasoning.

Amir Hooshangi (born 31 Jan 1989) Tehran, Iran. Bachelor of Science in software engineering 2006-2010 Karaj Payame Noor university of Iran. Member of Iran community of software and math researches. My major interests in researching are how to build the robust software in an efficient way which includes development teams, project management, software metrics etc. I'm currently working as senior J2EE developer and researcher in Iran telecommunication research center.