An Architectural Insight into the National Online Examination System

P Govind Raj, Pradeep Kumar, Soumya Sengupta, Kartikeye Vats, and P R Gupta

***Abstract*—Online examination system has been developed as per requirement of *NASSCOM* and *DIT*. The system can cater to a large number of students for administering multiple choice questions and True-False Questions of various subjects. System offers Dynamic paper generation using 3 parameter model based on Item response theory. *Flex, Spring and Hibernate* frameworks have been used for development of system and it is highly secure and fail safe. The system uses open source stack in its design and is extensible, reliable and scalable to handle future requirements. The system is further designed to support very large load using cloud computing paradigm. The paper presents the functional description and architectural design of the system along with Adaptive exam conduction model used.**

***Index Terms*—Online examination software, flex, spring, blazeds, terracotta, hibernate, eucalyptus, cloud computing.**

I. Introduction

Development of a National Online Examination System (*NOES*) within India was a *NASCOMM* recommendation which, Department of IT (*DIT*), Ministry of Communication and Information Technology is realizing through *C-DAC*. The purpose of such an endeavor was to design and develop a robust, fault tolerant, secure & scalable and an adaptive system, through which examinations can be delivered on an "on demand" basis in pre-identified examination centers spread across the country. The project is being carried out in two phases namely, Phase-1 wherein the software for conducting the exam was to be designed and developed and Phase -II wherein various colleges and partnering institutions are to be roped in to collaborate in providing questions to the question bank. Phase 1 of the project is nearing its complementation whereas Phase II has been initiated and is on-going.

The system utilizes various open source software framework. The integration of these frameworks to work as a cohesive unit has been one of the major engineering outcome of Phase 1 of the project. This paper presents an architectural perspective of the *NOES*. The paper also presents various issues faced as per design and implementation perspective.

The paper is divided into the 6 sections. Section II presents the functional and non functional requirements imposed on the system. Section III describes the System Architecture. It also provides the mapping of the non functional requirements to the architecture. Section IV presents key problems faced during implementation and their solutions.. Section V includes conclusion and future work planned.

Manuscript received October 9, 2011; revised March 5, 2012.

P. G. Raj, P. Kumar, S. Sengupta, K. Vats, P. R. Gupta are with Center for Development of Advanced Computing , NOIDA, India (email: pgovindraj, pradeepkumar@cdac.in, soumyasengupta@cdac.in, kartikyevats@cdac.in, poonamgupta@cdac.in).

II. Requirements of Examination System

Online examination system has various user centric provisions and administrative functions. The former include registration of candidates, selection of center, selecting examination time and date, and payment of fee (either through Demand Draft or through Credit Card). The latter include examination setup , admit card generation, verification of fee receipt, question pool management, question paper matrix preparation, creating nodes for different subjects for adding questions, verification of questions for their technical correctness, tagging of questions under different subjects, etc. The Fig. 1 shows the generic process flow of the system.

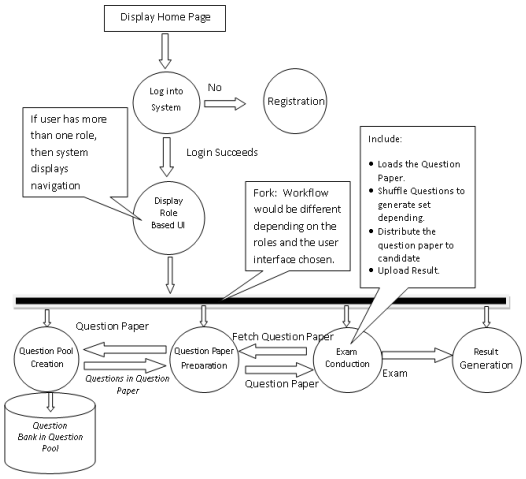

Fig. 1. Process flow diagram of online exam system.

Some key non functional requirements that had a strong impact on the architecture and the components used in the architecture includes: Open Source Stack to avoid vendor locking, Scale out Vs Scale Up, Ease of Extensibility, Performance/High System Throughput, Reliability, Security/Privacy/Escrow Accounts etc . All these functional and non-functional requirements have been considered, while deciding on technology components in the solution, described in further sections.

III. System Architecture

The *NOES* utilizes a number of open source frameworks at various tiers, namely, *Adobe Flex*[1] at the Presentation Tier , *Blaze DS*[2] at the Remoting tier, *Spring*[3] at the business tier, *Jasypt*[4] at the business tier, *Hibernate*[5] at the Object Relational Mapping Tier and *Terracotta*[6] for providing

JVM Level Clustering for High Availability and better throughput. System uses *Eucalyptus* for providing cloud computing services such as elastic scaling required for handling varying system loads. Fig 2 shows system components of online examination software at different tiers.

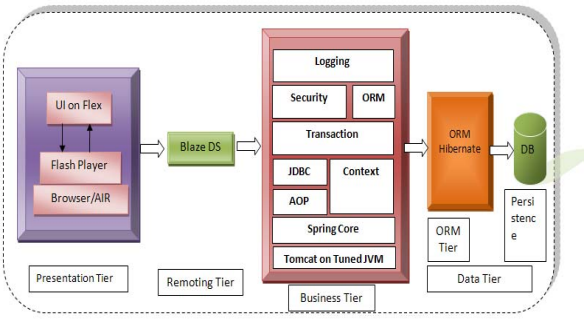

Fig. 2. System components in online examination software.

A. Presentation Tier

The software interface is designed and implemented as a Rich Internet Application (*RIA*). *RIA* is a web based application that has most of the characteristics of desktop applications. They are typically delivered by standard web browser plug-ins or independently via sandboxes or virtual machines. RIA has interactivity features which are often difficult to implement using normal HTML based applications. *Adobe Flex*, a framework for designing and developing *RIA*s, was used at the Presentation tier. By using *Flex*, a state full application running on the browser was developed for activities like student registration, payment of fees through online payment gateway, etc.

Use of *Flex* as a Framework also helps in creating HTTP based applications that run on the user desktop as opposed to the conventional browser. This was required in situations where in a complete "control" of Desktop was required e.g. when a user is giving an examination using the software; he should not be allowed to interact with browsers and any other applications. *Adobe AIR* framework helped in realizing such requirements.

B. Business Tier

One of the major considerations while designing the system was to ensure a design, agnostic to underlying Application Server. Enterprise Application Technologies like Enterprise Java Beans (*EJB*), although based on Java, are required to be coded to ensure portability across various applications servers of different vendors. Moreover, *EJB*'s are considered to be heavy weight as compared to a Plain Old Java Object (*POJO*).

Our business tier design uses *Spring* Framework to utilize the simplicity of *POJO* programming and at the same time relieves the application developer from concerns about issues like Transaction, Persistence etc. These "concerns" are managed by Spring Framework itself. Apart from being a replacement to *EJB*, *Spring* also provides aspect orientation to avoid code dangling and to provide modularization of cross-cutting concerns such as logging of users, Role Based Access Control [7] and Transaction management.

Declarative transaction management in *Spring*, separates transaction management code from the business methods via declarations and uses Transactions as Aspect. Various transactional attributes like propagation behavior, isolation level, rollback rules, transaction timeout etc can be specified using annotations. This helps programmer in concentrating more on developing functional requirements in place of transaction management aspects. However, the development team faced issues while implementing transaction management as discussed later in section IV.

Security was one of the key non functional requirements of the *NOES*. The security subsystem of the *NOES* provides Authorization, Role Based Access Control (*RBAC*) and cryptographic functions for storing user credentials for authentication; through Spring Security. For providing cryptographic functions like encryption Java Simplified Encryption (*Jasypt*) was integrated with *Spring* Security and *Hibernate*, so that *Spring* Security uses password encoders etc provided by *Jasypt*.

Using the Security Subsystem, access control can be provided at the *URL* level, Method Invocation Level, domain object level and view rendering level. The authorization is implemented with the help of *Spring* Security as a set of Intercepting Filters as shown in Fig. 3.

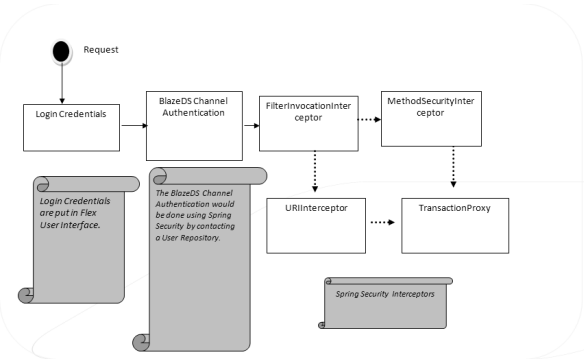

Fig. 3. Intercepting filters for authentication.

For making Access Control Decisions the *Spring* Security Access Decision manager were used. All these access decision managers use a group of voters to be configured for voting on access control decisions. *NOES* utilize Role Voters and Authenticated Voters. Role Voters votes for an access control decision based on a user's role.

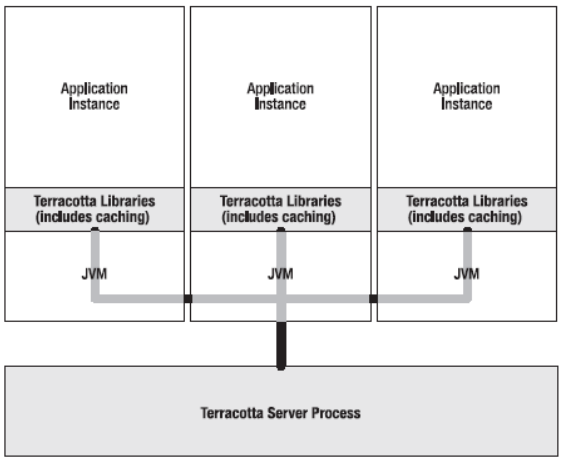

Fig. 4. Terracotta running application instances.

The Fig. 4 shows applications running on top of Terracotta. Using *Terracotta*, application is clustered at the JVM level for high availability and scalability [8], [9]. *Terracotta* keeps objects with cluster level scope instead of JVM level scope. Since objects are now available across the cluster, there are no single points of failures. Further, *Terracotta* has been configured in such a way that it persist the object graph. This helps to recover even in situations wherein there is a full cluster restart.

Using *Terracotta* also allowed us to scale out the application by just increasing the number of *JVM*'s in the setup. Therefore, transparent scaling was achieved without requiring a change in source code for the purpose of scaling.

Terracotta is also used to implement various design patterns like Write behind to System of Records (*SOR*), Asynchronous Commit etc with intent to minimize Database I/O during an examination. The usage of these design patterns are mentioned in TABLE I.

TABLE I : USE OF KEY DESIGN PATTERNS IN NOES

Design Pattern used	Use in NOES
Model View Control	Separation of Presentation Tier with Business Tier
Proxy Pattern	Used for implementation of Advices when implementing Aspect Orientation in Business Tier

Terracotta also provides a transparent *Hibernate* 2nd Level Caching which increases the throughput of *NOES* and further reduces Disk I/O. This results in improving the performance by decreasing the load on the system. The deployment architecture of the system is shown in Fig. 5. The figure shows *DNS* based load balanced (*LB*) cluster of *Apache mod_proxy* based load balancer interacting with *Tomcat Servers* (*TC*). *TC* are clustered at their *JVM* by *Terracotta* Mirrored Server Arrays(Tera1.. n). *Eucalyptus* is being used as a cloud platform for providing Elastic Scaling. Whenever the load is increased beyond a threshold limit, Eucalyptus is instructed to spawn, required number of *Xen* Virtual Machines on available Nodes. Thereafter, a specific Eucalyptus Machine Image (*Emi*) is deployed on top of *Xen* to run a particular machine instance.

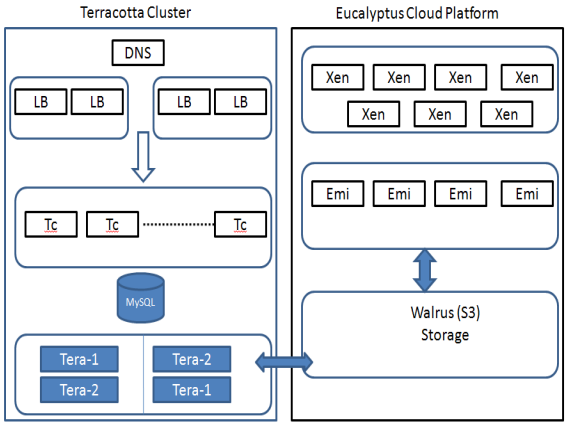

Fig. 5. Deployment Architecture.

TABLE II shows mapping of non-functional requirements of system with various technology components used in the design. TABLE III shows the results of load testing run of system along with setup details used. A total of two *TC* instances were used to perform load testing.

TABLE II : MAPPING OF NON-FUNCTIONAL REQUIREMENTS TO VARIOUS TECHNICAL COMPONENTS

Requirements	Flex	Spring	Jasypt	Spring Security	Terracotta	Eucalyptus
Scalability	√	√	-	-	√	√
OSS Stack	√	√	√	√	√	√
Expandability	-	√	-	-		
Transparent Scale Out	-	-	-	-	√	√
High Availability	-	-	-	-	√	√
High Volume Transaction	-	-	-	-	√	-
Security	-	-	√	√	-	-

IV. ISSUES FACED DURING IMPLEMENTATION

A. Integrating Flex and Spring

Flex can access java object and *Spring* beans are nothing but java objects therefore ideally integration of *Flex* and *Spring* should be simple. However, whenever *Flex* contacts *Spring* business tier using *BlazeDS*, *BlazeDS* instantiates Java Objects. This default behavior does not fit our architecture as *Spring* should instantiate the components and inject these components whenever needed.. The key to the *Flex/Spring* integration, therefore, is to configure *BlazeDS* to let the *Spring* container take care of instantiating Spring beans by the use of suitable factory classes provided by *BlazeDS*. In the first version of *NOES* , we designed and developed a *SpringFactory* and configured *BlazeDS* to use this factory class to let *Spring* instantiate the classes and perform dependency injection when needed. Later the *Spring* Community came up with *Spring BlazeDS* Integration Components [10]. The current version of *NOES* uses this new scheme to integrate *Flex* and *Spring*.

B. Integrating BlazeDS Channel Security and Spring Security

BlazeDS has got methods to authenticate Channel sets between *Flex* Client and the Server Side Code. *NOES*, however, use *Spring* Security to implement security functions. This again required that *BlazeDS* should allow *Spring* Security to manage security. For this purpose, we extended the functionalities of *BlazeDS* and designed and developed a *Login Command Class* which allowed, *BlazeDS* to use *Spring* Security functionalities. Later when *Spring* Integration Components [11] were released, we shifted our code base to use it.

C. Issue Faced during Implementing Transaction Management using Spring.

NOES use *MySQL* as a database server at present (though the design is agnostic to the database at the backend). The default database engine of MySQL in Linux is *MyISAM* [12]. Transaction management provided by *Spring* does not work

with *MyISAM*. The solution to this problem was to change the database engine from *MyISAM* to *InnoDB*[13].

Spring uses the Proxy Pattern to implement Aspect based Transaction management. Therefore, methods called using *Proxy* were only advised for transaction management. In our service class there were situations when a proxy method (*f1*) called another method (*f2*). Java calls method *f2* using "*this*" pointer. In such cases the method *f2* is not advised for transaction management. Therefore if , some exception happened while execution was in *f2* , it had to be manually handled.

To work around, we have taken a reference of the interface type implemented by the service and created a bean post processor which assigns the proxy to the reference. This is called self reference. Now whenever, a function is called inside a method which is advised for transaction, self reference is used instead of *this* pointer.

NOES sends automated E-mails to candidates triggering a successful completion of an event such as registration successfully done, admit cards generated etc. Most of these situations involve a database operation to either Insert, Update or Delete records. Such methods are required to advise as a Transactional method. In such scenarios, the sending of email is typically the last step after a successful database operation. However, *Spring* buffers the database operation and performs it at the end of the Transactional Method i.e. after sending the mail. This can lead to a situation where in the database operation was not successful, however, the e-mail was sent. To get around this issue, we flush *Hibernate* session right after issuing a command to save, update or delete an object. Doing this throws an exception immediately on failure of a Database operation and thus prevents sending of mails in case of exceptions.

Whenever a Transactional method is called from the *Flex* end and it fails, nothing was returned from the Business tier to the presentation tier and the transaction was not rolled back. The reason for this was that, Transactional method did not throw out an exception to the calling method and the *Spring* Transaction manager only rolls back an exception if the transactional method throws out an exception. To solve this issue, a service method was introduced such that it was not treated as transactional and this method was exposed to the presentation tier. The service method internally called a Transactional method. This transactional method did not handle exceptions and threw out exception to the service method. In this case, *Spring* Transaction Manager rolls back the Transaction and the service method can be used to handle the exception and send relevant message to the presentation tier.

D. Integrating Flex and Terracotta

Terracotta has defined several classes which it treats as non-portable. These non-portable classes cannot be used in clustered mode across different *JVM*'s. *Flex* uses some such classes pertaining to *HTTPServlets* in its messaging session thereby making clustering *Flex* Sessions using Terracotta incompatible. Since the application at the front end was a state-full one, reliance on sessions was not necessary and therefore *Terracotta* was configured to ignore non-portable classes which were used by *Flex* in its messaging session. Further, we had used a load balancer based on *Apache* to use sticky sessions. This allowed *HTTP* request to be routed to the same application server where it was first originated, this helped in non-sharing of sessions across the cluster. In case of a node failure, the user just needs to re-login and he can continue from where had left as since his progress is stored across cluster using *Terracotta*.

E. Concurrency Issues with Terracotta Asynchronous Committer

Terracotta AsynCommitter allows to commit records in a asynchronous manner through configuration of processing buckets, present at each application server. When two or more processing buckets tries to commit a data element, at the same time and the primary key of such a data element is a numeric increment , then there will be Data Integrity Violations as two data elements may have the same number as primary key. Further, the application can enter into an infinite loop as in case of an error while committing in the database, *Terracotta* tries to recommit the values. While recommitting again the same situation of Data Integrity may arise. This situation can go on till a serial schedule is found. To overcome this issue , we have used *UUID* as the primary key of data items which are stored asynchronously. This prevents Data Integrity Violations in asynchronous commit mode.

V. Implementation Adaptive Assessment System in NOES

Item Response Theory (*IRT*) provides the theoretical framework for implementation of Adaptive Testing. The *IRT* algorithm aims to provide information about the functional relation between the estimate of the learner's proficiency in a concept and the likelihood that the learner will give the correct answer to a given question. The amount of knowledge, learning ability, proficiency in a subject of a person, etc., cannot be directly measured like height or weight. These are generally referred to as the latent traits or "ability" in IRT [14]. The aim of *IRT* is to estimate this ability. *IRT* rests on the postulate that an examine has a definite probability of giving a correct answer to the given question, and this probability will be high for high ability examinees and low for low-ability ones.

NOES test an examinee on multiple subjects like Aptitude, Programming Language, Data Structures, etc. Therefore, the question paper for the examinee needs to have questions from all these sections. Further, each section must have enough questions of various difficulty levels so as to precisely determine the ability of examinee. For the recruitment exam conducted using *NOES*, there were 12 sections for each examinee. Being adaptive in nature, each section had to contain 30 questions from each of 5 difficulty levels where 30 is the upper limit of the number of questions that an examinee can attempt in a section. So a single question paper can contained 30*5*12 or 1800 questions. Major issue was to send such a huge set of questions to hundreds of examinees simultaneously at the start of the exam. In order to reduce the load on the network, the questions were sent to the examinees only section wise.

Another design issue relates to calculating of the ability estimates of the examinees and administering the next best question corresponding to his/her ability. Ability estimation of an examine, for a section, takes into account all the responses made by the examinee in that section. This involves a huge amount of calculations as ability estimation and question selection has to be done after each response for each examinee as the process is iterative. In order to relieve server, from the workload of the above calculations, it was decided that the calculations involved for an examinee must be done at the client machine which the examinee is using. RIA client capable of working outside the browser environment helped us in realizing it [15].

At the start of each section the client gets all the *question ids* corresponding to that section. The client calculates examinee ability , decides the difficulty level of the question to be given next and then sends to server, the *question id* of the calculated difficulty level. On receiving the *id* the server sends the question along with its answers and the correct option. The examinee sees the question text along with the answer options and makes a response. TABLE III shows calculation being done in a session for a candidate based on the responses. *AAS* test can be stopped if any of the following criteria are met.

TABLE III : SETUP DESCRIPTION AND LOAD TESTING RESULTS

Setup Description	Results of load testing
• **Apache Load Balancer**: • **Tomcat** : Max Thread 1500 (One Tomcat instance), Min Spare Thread : 100 • **JVM** : Initial 256 MB , Max 2GB , Chunk Size 256 MB • Peak size Reached during load test 400 MB • **MySQL** : max_connections=6000 • **Distributed 2nd Level Cache** • Caching Policy : Non Restrict Read Write	• Max Load Tested: 5000 • Ramp up Time: 50 sec • Idle Time: 5 sec [Presumed Thinking Time] • Peak CPU Usage: 12 % Response Time: • Cumulative Avg: 0.9155 Sec • Login/Fetch QP: 0.814 sec • SetAnswer: 0.087 sec • Clock sync: 0.0075 sec • EndExam: 0.0070 sec

- The ability of an examine has been calculated with desired precision (i.e., the standard error in calculating the ability has fallen below a predefined value).
- The first 'n' numbers of questions is consecutively answered correctly or wrongly. The examinee is given the highest/lowest ability for that subject or examination based on whether all answers are correct or incorrect.
- The maximum number of questions for a particular section or the whole of examination has been administered to the examinee. This case may arise when the computer is unable to precisely predict the ability of examinee.
- The maximum time limit for a particular section in an examination or the total time limit of the examination has been reached.
- The item bank has got exhausted which may be the case when item bank is quite small.

VI. CONCLUSION AND FUTURE WORK

Using the NOES, C-DAC has conducted three PG Entrance Examination and two national level recruitment examination successfully. The question pool of NOES has 97,000 questions on Electronics, Computer Sc & engineering encompassing 400 different Topics along with questions on aptitude.

The National Level recruitment Examination was conducted across 10 centers of C-DAC for a period of 10 days each day having 3 slots. Therefore, more than 500 live runs of the system are already conducted so far. DOEACC has also expressed its interest in conducting CCC examination.

The initial stress testing result shows that we can conduct an exam for 5,000 concurrent users using just 2 Tomcat server instances. We plan to scale the application to support around 1, 00, 000 concurrent users in a single session.

NOES uses Hibernate in its ORM Session. One of the criticism of ORM technologies, is that generates non-optimized SQL Queries for databases. We plan to tweak the ORM layer and use frameworks like *AutoFetch* to generate Optimized SQL.

ACKNOWLEDGMENT

Authors would like to thank Sh. GV Ragunathan Sr. Director DIT, Sh. Anil Pipal Addl Director, DIT for their constant motivation and support throughout the project life cycle. Authors also express their sincere thanks to Sri D K Jain , Sh R K Singh , Sh V K Sharma and Dr George Varkey for their constant support in executing the project .

Authors also express thanks to team members Uttam Kumar, Pankaj Nirwan , Kanti Singh, Neha Sharma and Vikram Vijh .

REFERENCES

[1] J. Balderson, P. Ent, J. Heider, T. Prekaski, T. Sugden, and A. Trice, "David Hassoun and Joe Berkovitz," *Professional Adobe Flex 3*, Wiley Publishing, 2009

[2] S. Tiwari, *Professional BlazeDS: Creating Rich Internet Applications with Flex and Java*, Wiley Publishing, 2009.

[3] G. Mak, K. Sipe, J. Long, and D. Rubio, *Spring Recipes: A Problem-Solution Approach*, Apress, 2009.

[4] Jasypt . www.jasypt.org, (Feb 19, 2012).

[5] C. Bauer and G. King, *Hibernate In Action*, Manning, 2004.

[6] Terracotta, *The Definitive Guide to Terracotta: Cluster the JVM for Spring, Hibernate, and POJO Scalability*, Apress, 2008.

[7] P. Mularien, *Spring Security*, Packt Publishing, 2010.

[8] Terracotta. JMX Guide, (Feb 19, 2012). www.terracotta.org/confluence/display/docs/JMX+Guide

[9] Spring Clustering with Terracotta, (Feb 19, 2012). www.artima.com/lejava/articles/spring_clustering.html

[10] J.Wischusen, *Professional Cairngorm,* John Wiley & Sons, 2009.

[11] C. Giametta, *Pro Flex on Spring*, Apress, 2009.

[12] Vaswani, *Mysql: The Complete Reference*, TATA McGRAW HILL, 2004.

[13] B. Schwartz, P. Zaitsev, V. Tkachenko, J. D. Zawodny, A. Lentz and D. J. Balling, *High Performance MySQL*, O'REILLY, 2008.

[14] F. Baker, *The Basics of Item Response Theory*, ERIC Publications, 1985.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 2007.

Mr. P. Govind Raj is working as Senior Technical Officer in the Design Group at CDAC, Noida. He is coordinating the development activities of online examination system at CDAC, Noida. His research interest includes Ubiquitous computing, e-Security and Open Source Systems.

Mr. Pradeep Kumar did his M. Sc. in Informatics from Delhi University. Presently he is working as Technical Officer at CDAC, Noida and is involved in development in Spring Framework.

Mr. Soumya Sen Gupta is working as Technical Officer at CDAC, Noida. He is currently with the Design Group at CDAC, Noida. His areas of interest include Spring Framework and application security.

Mr. Kartikye Vats completed his B.Tech(I.T) from Jaypee Institute of Information Technology. He worked as a Configuration controller in Infosys till Oct 2008. He is currently working as Flex developer.

Dr. P.R. Gupta has more than 20 years of experience in academics and research. She has M.Tech from IIT Delhi and Ph.D. in Computer Sc. & Engg from KNIT, Sultanpur. Presently, she is working as Professor and Head of School of IT at CDAC, Noida. Her research interest includes Ubiquitous computing, Artificial Intelligence, information security, Open Source Systems, e-governance and IPR issues.