Game operation query language for facilitating game server’s FCAPS operation

Hwan-Soo Yoo and Seong-Whan Kim*

School of Computer Science,
University of Seoul,
163 Seoul Siripdae-ro,
Dongdaemoon-gu, South Korea
Email: rayryu7@uos.ac.kr
Email: skwim7@uos.ac.kr
*Corresponding author

Abstract: Game companies require successful game operation for their revenue maximisation. In an operation phase, they need not only operators who are experts in non-functional and operational issues but also developers who are experts in game development. Game server should manage several operational issues such as fault recovery, performance metering, and configuration management during game service period. However, most game server operators do not have programming expertise. To support successful fault, accounting, configuration, performance, security (FACPS) operation without programming expertise or development knowledge, we introduce domain specific language (DSL) operation approach for usual game operators. We design GOQLDSL, focusing on fault, configuration, and performance management issues. We experimented with 15 game server operators who are working in major game publishers. Experimental results show that GOQL was attractive to game operators because it is functional and does not require programming knowledge even in the situation that needs programming source modifications. For the future research, we will extend our textual version of GOQL to visual language version to give more convenience and intuitive advantage to game operators and novice developers. To cover this, we require study on visual notations, UX-related elements, and visual transformation of text.

Keywords: game server operation; game operation; game server; fault, configuration, accounting, performance, and security; FACPS; domain specific language; DSL; aspect-oriented programming; AOP; development operations; DevOps; aspect; reliability, availability and survivability; RAS; game operation query language; GOQL.


Biographical notes: Hwan-Soo Yoo is a PhD candidate in School of Computer Science at the University of Seoul, South Korea. From 1998 to 2008, he developed an internet browser, a short message system centre (SMSC) server, a location-based system centre (LBSC) server, and a mobile game platform for major telecommunication companies. His current research focuses on web-based multimedia authoring system, and multimedia system development using domain specific language.

Copyright © 2018 Inderscience Enterprises Ltd.
Seong-Whan Kim received his PhD in EECS from the KAIST, Taejon in 1999. He worked for IMT-2000 infrastructure division as a software team leader in LG Electronics from 1997 to 2000, and was responsible for wireless LAN infrastructure design as a Senior Software Engineer in Cisco Systems Inc. at San Jose from 2000 to 2001. He is currently a Full Professor at the School of Computer Science, University of Seoul, Seoul, Korea. Since 2015, he has worked for the Ministry of Culture, Sports, and Tourism as the National R&D Director. His research interests include mobile communication systems, mathematical modeling of human visual system, and its application to multimedia coding/restoration/transmission.

This paper is a revised and expanded version of a paper entitled ‘GOQL: game operational query language for facilitating game server operation’ presented at International Conference on Platform Technology and Service (PlatCon-15), 26–28 January 2015, International Convention Center Jeju (ICC JEJU), Korea.

1 Introduction

As games have been put into the spotlight as a promising source of profit, an increasing amount of companies have started to develop games that are now easily downloadable; thanks to the currently invigorating App stores (App Store, n.d.). Development organisations are now adopting development methods such as model-driven development (MDD) (Atkinson and Kuhne, 2003), software product line (SPL) (Paul and Linda, 2002), domain-specific language (Van Deursen et al., 2000), and aspect-oriented programming (AOP) for effective development (Elrad et al., 2001). In addition, due to the irrelevance of prevalent concept of development and operation, as a way to rid of this misconception, and for the cause of effective development and operation, concepts such as development operations (DevOps) (Hüttermann, 2012) are emerging to contribute to harmoniously combining the developer with the operator. In spite of such effort, there are number of problems related to game development as shown below. For example:

1. the developer does not develop products optimised for operation
2. alteration on the source level developer made is required once operated
3. the lack of exclusive tools or languages necessary in operating game servers is to be taken into consideration.

As a solution for such problems, we present an operational domain-specific language, which we call game operation query language (GOQL). Users can utilise GOQL for FCAPS operation on game management without programming expertise knowledge. We focus more on fault, configuration, and performance management issues than on accounting and security issues. The rest of this paper is organised as follows. Section 2 reviews related works on domain specific language (DSL) and AOP (Kiczales, 1997). Section 3 shows our methodology. Section 4 shows experimental results. We conclude in Section 5.
2 Related works

In this section, we briefly review about previous works using DSL, MDD, AOP and FCAPS (FCAPS and TMN, n.d.) model. DSLs are languages that define the jargon of a particular class of problem domains or set of domain aspects. Executable DSLs hide software implementation (Stappers et al., 2012). DSL is referred to as either DSL or just simply domain language but is often referred to as DSL by academic community. Terminology is in contrast to general languages such as C or Java language and general modelling language such as UML (Selic, 2005). It is a limited language that is concentrated to specific problem domains by appropriate abstraction and notations. It is a problem-centric language (Langlois et al., 2007) built for one particular task in mind. Development teams use same vocabulary to represent program module. For example, if there is a requirement for bond trading module within problem domain, same vocabulary is used during code writing [Ghosh, (2011), p.8] How to integrate domain-specific languages into the game development process (Walter and Masuch, 2011) suggests DSL as a tool for supporting tasks from game design to implementation.

A DSL-based approach to software development and deployment on cloud (Sledziewski et al., 2010) providing a method for bridging the gap between the clients view and software development on the cloud is proposed. It integrated DSL into the process of cloud-based application development and deployment. It is comprised of three main components which are the model designer, the code generator and the deployment tool. These however do not portray DSL as properly made and put into use, but show only the screenshot of it.Neptune (Bunch, 2013; Bunch et al., 2011) provided DSL that automates configuration and deployment of existing high performance computer (HPC) (Dongarra et al., 2005) software via cloud computing platforms. It integrated Neptune into a popular, open-source cloud platform, and extend the platform with support of user-level and automated placement of cloud services and HPC components. Figure 1 shows the code of Neptune, which is the sample code that determines the amount of the message passing interface (MPI) (‘MPI (Message Passing Interface (MPI) | High Performance Computing Virtual Laboratory, n.d.) nodes and its results.

Figure 1 Typical format of a user’s Neptune code from Bunch (2013)

```ruby
result = neptune :type =>:mpi,
  :code => '/code/powermethod',
  :nodestouse => 4
if result [: success]
  puts 'Your MPI job is now in progress.'
else
  puts 'Your MPI job failed to start.'
end
```

An aspect-oriented model driven framework (Simmonds et al., 2005) provided framework combining AOP and MDD, and also proposed the use of concern modelling languages to adopt the model drive architecture (MDA) approach. The primary focus of the framework is the transformation of aspect-oriented models from more abstract forms
to more detailed forms. The major activities are partitioned into four categories: source level, mappings, target level and model composition. The source level includes activities acquiring or developing abstract aspect and primary models. The primary model is developed by the system architect. The system architect decides what features will be included in the primary model and which will be treated as aspects. The mappings category includes activities developing or acquiring the corresponding target mappings for the aspect and primary models. The model composition part includes activities instantiating and composing the aspect and primary models using bindings. Although this paper is on the novel approach of feature separation through the combination of AOP and MDD, our thesis is applicable to new development and legacy system unlike others that are applicable to new development only.

AOP is a software development paradigm developed for the purpose to overcome the limitations of object-oriented programming. There are inevitable concerns on various objects that need to be taken care of within a single class in terms of existing object-oriented programming. For instance, in the case of a certain class that must handle business logic, not only the core business logic but also the exception handing of database connection, the logging of transaction, and the user authentication logic are all tangled up within the source code. The concept of dividing each of the logics which are concerns into different classes of modules is known as AOP. Such tangled source codes as seen above goes against the single responsibility principle (SRP) (Martin, 2003) and open/close principle (Martin, 2003). AOP consists of the following fundamental concepts.

- **Concern:** Various logics demanded within a program.
- **Core concerns:** core business logic.
- **Crosscutting concerns:** logging, authentication, database connection.
- **Aspect:** separation of crosscutting concerns by introducing a new unit of modularisation.
- **Join point:** The system exposes points during the execution of the system. These may include execution of methods, creation of objects, or throwing of exceptions. Such identifiable points in the system are called join points (Laddad, 2009).
- **Pointcut:** A mechanism for selecting joint point.
- **Join point model:** Implementing a crosscutting concern requires selecting a specific set of join points. The concept of join points and the point cut construct together form an AOP system’s join point model (Laddad, 2009).
- **Weaving:** the action of composing the final system.
- **Advice:** source codes that are inserted into existing source codes during dynamical run-time.

AOP goes through the implementation of the extension of various major languages, and the major language and AOP extension such as AspectJ (Laddad, 2009), AspectC++, and Post# are most frequently used. AOP4CSM (Mdhaffar et al., 2011) proposed AOP4CSM, is based on AOP and monitors quality-of-service parameters of the software-as-a-service
Game operation query language for facilitating game server’s FCAPS layer. It intercepts client and server methods at well-defined join points to collect data at important instants of time. These instants are:

1. $t_1$ is the instant when the client invokes the request
2. $t_2$ is the instant when the server receives the request
3. $t_3$ is the instant when the server sends the response
4. $t_4$ is the instant when the client receives the response.

This proposed aspect code computes the number of successful invocations (processed requests) while intercepting the corresponding method at the join point that corresponds to $t_4$. Moreover, the number of all sent requests are evaluated by advice 1.

‘Security auditing method based on aspect oriented programming (AOP) and annotation information system’ (Google Patents, 2014) provided the combination of the AOP and an annotation technology. It claimed that the method solves the problem of decoupling of security audit and business, and can provide humanised safety audit information. Aspect-oriented system monitoring and tracing (Lopes et al., 2002) provided method about an aspect which encapsulates the monitoring/tracing behaviour. This behaviour may transparently be forced onto the resource by compiling the object class for the resource along with the monitoring/tracing aspect. It claims that when the monitoring/tracing is no longer needed, it is removed simply by recompiling the resource object classes without the aspect. Monitoring advice goes through weaving in each of the database servers, printer servers, web servers, and document servers.

FCAPS (Rec, 2000) is the ISO telecommunications management network model and framework for network management. FCAPS is an acronym for fault, configuration, accounting, performance, security, the management categories into which the ISO model defines network management tasks (FCAPS – Wikipedia, The Free Encyclopedia, n.d.). A majority of the following explanation of FCAPS has been excerpted from (Rec, 2000).

- **Fault management**: Fault management is a set of functions which enables the detection, isolation and correction of abnormal operation of the telecommunication network and its environment. It provides facilities for the performance of the maintenance phases from ITU-T M.20. The quality assurance measurements for fault management include component measurements for reliability, availability and survivability (RAS).

- **Configuration management**: Configuration management provides functions to exercise control over, identify, collect data and provide data to network equipment (NE). Configuration management supports the following function set groups; network planning and engineering, installation, service planning and negotiation, provisioning and status and control.

- **Accounting management**: Accounting management enables the measurement of the use of network services and the determination of costs to the service provider and charges the customer for such use. It also supports the determination of prices for services. Accounting management includes the following function set groups; usage measurement, tariffing/pricing, collections and finance, and enterprise control.
• Performance management: Performance management provides functions to evaluate and report upon the behaviour of telecommunication equipment and the effectiveness of the network or network element. Its role is to gather and analyse statistical data which is for the purpose of monitoring and correcting the behaviour and effectiveness of the network, NEs or other equipment and also it is to aid in planning, provisioning maintenance and the measurement of quality. As such, it is carrying out the performance measurement phase of ITU-T M.20. A TMN collects quality of service (QOS) data from NEs and supports improvements in QOS. The TMN may request QOS data reports to be sent from the NE, or such a report may be sent automatically on a scheduled or exception basis. At any time, the TMN may modify the current schedule and/or exception thresholds. Reports from the NE on QOS data may consist raw data (data gathered in the course of providing telecommunication services) that is then analysed externally to the NE, or the NE may be capable of carrying out part of the analysis of the data before the report is sent.

• Security management: Security management provides the management of security. In addition, security of management is required for all management functional areas and for all TMN transactions. Security of management appears as part of the security function in ITU-T M.3010. Security of management functionality includes security services for communications, Security event detection and reporting. Security management includes the following function set groups; prevention, detection, containment and recovery, security administration.

3 Our approach

3.1 GOQL model

Game server faces operational issue such as fault, performance and configuration during the service of game. But it is hard to find that game server operators know programming expertise. We introduce game server operational language based on DSL approach. By using it, operators can easily operate servers without programming expertise. To do so, we adapted FCAPS that focus on operational issues of telecom companies. Our model shown in Figure 2 reflects FCAPS aspects for facing game server operational issues. It handles following issues:

1 fault: fault detection and alarm
2 configuration: configuration of network topology, server instance, etc.
3 accounting: managing user, item, payment, device, push and batch information
4 performance: managing server instances, processes, network and databases
5 security: managing log and security breach.

DSL model of GOQL is shown in Figure 3. It reflects management facet of FCAPS. A GOQL app is a 6-tuple, \(\{S, C, V, E, A, P\}\), consisting of a set of servers \(S\), a set of management facet called collection \(C \in S\), a set of granular management facet called view \(V \in C\), a set of realised view called entity \(S \times C \times V \rightarrow E\), a set of behaviour on
entity called action \((E \times A) \rightarrow E'\), and a set of manageable data called property \((name, value) \leftrightarrow P\).

**Figure 2** FCAPS enacted model of GOQL (see online version for colours)

**Figure 3** DSL model of GOQL

In this paper, we define App as an application. The application acts as game with more than one server is required to execute. We define \(\{S, C, V, E, A, P\}\) as follows:

- **Server**: A server is a term that mainly refer to login, session, game server, etc. as we have mentioned before.

- **Collection**: A server consists of more than one collection. A collection refers to the artefacts that are relevant to each of the management facets of FCAPS.
• **View**: A view refers to the subordinate collections within a collection, which is also the minimal unit of management. As an example of the accounting management mentioned above, the board, broadcast, private and poi of a push collections can be seen as a view.

• **Entity**: A view is realised as an entity. At this point, a view may be conceived as a class of an object-oriented concept, and an entity as an object.

• **Action**: When an entity is created, it includes an action. This refers to the simplest form of behaviour of an operation.

• **Property**: The property refers to the most atomic artefacts of our model when it acts.

3.2 **GOQL Workflow**

We propose operational methods using the mentioned DSL model. **DSL workflow** hosts all activities that focus on the operation and development of a server. This workflow results in the production of DSL. **DSL workflow** is consist of identification, DSL-isation and development and operation.

3.2.1 **DSL workflow: Identification**

In this phase, only small efforts are spent identifying reusable artefacts, such as source code and documents. Not everything in the game domain can be identified at this phase. When there are existing usable source and documents, they go through identification so that they are accessible in the specification process during the next phase.

3.2.2 **DSL workflow**

DSL-isation: This phase consists of three steps:

1. **DSL artefact specification**: Fitting operational method for DSL artefacts specification is made by combining artefacts that are produced by previous identification phase and operational workflow at identification phase. Method artefacts such as server, collection, view and action are identified at this phase.

2. **Operational aspect generation**: This is the step in which aspect is applied. In this step, the method of applying operational aspect to legacy code, and the method of applying the aspect when developing new games will be dealt with. When applying to the legacy servers, the developer can revise and apply the code in person, or semi-automatically. The details of those procedures are shown in Table 1.

3. **DSL implementation**: the stage in which DSL is produced. At this stage the DSL which is used by the operator is produced. This production of DSL is achieved through the use of aspect as defined beforehand to lighten the burden of the developer, and activates the reuse of the existing code. At this point, DSL is produced through internal or external methods determined by the programming level of the operator. In our method, however, we encourage the use of high level expressiveness external DSL.
### Table 1  Methods of applying operational aspects

<table>
<thead>
<tr>
<th>Server type</th>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy</td>
<td>Manually</td>
<td>The developer must apply each and every one of them by hand.</td>
</tr>
<tr>
<td></td>
<td>Semi-automatically</td>
<td>The aspect application algorithm is applied.</td>
</tr>
<tr>
<td>New</td>
<td>Manually</td>
<td>The operational aspect is developed after it is confirmed and applied by the developer when developing new servers.</td>
</tr>
</tbody>
</table>

#### 3.2.3 DSL workflow: Development and operation

This is the phase in which the developer and operators actually operate the game. Operators may request new functions needed for game operation to the developer and then, the developer may include new collections, views, entities, actions, properties, etc. to the DSL based on such requests. When a new operational aspect occurs, it leads back to the previous step of DSL implementation and iterates.

The operational workflow hosts fault, configuration, accounting, and performance and security management aspects. The operational method is based on principles that are iterative and related to incremental development, software reuse, DSL, AOP and DevOps. The operational workflow consists of two steps:

1. **Operational workflow: Identification**: Six criteria of fault, configuration, accounting, performance, and security are identified within the server. Similar to the DSL workflow, since this is not the final phase, only identifiable criteria go through this process. Important criteria such as servers should be identified.

2. **DSL-isation**: The artefacts defined in the operational model such as collection, view, entity, action, and property are specified at this stage. These specifications are added to the artefact specification of DSL workflow. At this stage, the DSL obtains syntax and semantics appropriate for its company.

#### 3.3 GOQL language

The launching of games is easily accessible through app stores and development tools such as Unity (Unity – Game Engine, n.d.) (which support multi-platforms such as iOS, Android, Windows, Mobiles, and PCs) and it has also contributed to this increase of games. Game development start-ups do not begin business with enough developers, so it is difficult for the developers to concentrate fully on the development and operate as well during the phase after the game launching and incoming of the revenue. However, not only such does operation procedures result problems in revising initial bugs in applications, but has a negative influence on that they have to continue to maintain their high ranking positions in app stores by applying the users’ needs instantly. As a solution to such problems, operators have to come up with an operational language that can be utilised without the help of the developers. As mentioned before, the problems related to the operational process can be added up to the realm of FCAPS.

We design the language as below, using our GOQL model. It will be designed so that it maximises expressiveness, and is simple so that it is easily accessible to operators without programming knowledge. Language structure is shown in Figure 4. `<server>` part is for designating server type such as login, lobby, game and session servers.
<collection> part is for designating fault, connection, storage, push, etc. <view> part designates granular management artefact of server. Entity is created by using above <server><collection><view> parts. <do> part designates multiple <action> parts. It sets behaviours on entity. <property> designates manageable data.

**Figure 4** Basic language form

\[
\text{COQL ::= } <\text{server}> <\text{collection}> <\text{view}> <\text{do}> <\text{action} <\text{property}>>\]

Figure 5 shows the sample language statement about fault management of connection: detect/alarm (#5 and #6 are shown the action statements). Reading and updating user information are shown in Figure 6. It reads user ‘John Doe’ and update address information.

**Figure 5** Fault management of connection: detect/alarm (see online version for colours)

1. server: login
2. collection: fault
3. view: connection
4. do
5. detect 'latency': '1s' and alarm
6. detect 'connection': 'failure' and alarm

**Figure 6** Accounting management of user: read/update (see online version for colours)

1. server: user
2. collection: userInfo
3. view: user
4. do
5. read 'user': ‘John Doe’
6. print
7. update 'address': 'Korea,Seoul,Kangam-gu,Daechi-dong, 1005'

### 4 Experiments

To evaluate the performance of operational aspect, we have conducted several experiments in a real game server environment. Game server is comprised of environment shown below in Figure 7. The red numbers indicate the port number of servers connected with each other, while the green indicate the number of server instance within server farm. This game is currently in service, and the number of simultaneously connected users reach approximately 500,000. The total number of subscribed users is 5,000,000. Server farm consists many servers. The login server farm is made of one server, while lobby server farms consist of two. The game server farm is made of two servers, while push server consist of two. These servers are not physical servers within the company, and they are configured by Korea Telecom (KT) cloud (ucloud biz, n.d.). When during the experiment of the addition and deletion of server instance within a
particular farm, KT cloud provides API which easily enables these tasks. By using this application programming interface (API), we simplified the job.

**Figure 7** Evaluation test environment: server topology (see online version for colours)

4.1 Overhead of operational aspect

The main objective of this first category of experiments is evaluating the overhead of operational aspect. Doing this, experiments are conducted by two cases. These are game server with operational aspect and legacy game servers without operational aspect. Each test dealt with the operational aspect of fault, configuration, accounting, performance, and security. By modifying our server source, we measured the response time of specific jobs. In the game server with operational aspect, we inserted timestamp at the beginning and end of each operational aspect. In the legacy server, the timestamp of join point was inserted into each of the operational aspect of the same location.

- *The performance of fault operational aspect:* This was conducted ten times, and the source was modified so that the push server was intentionally shut down. The main server was modified to connect push server last. Legacy server was modified so that it printed only exceptions on the screen. In the case of the aspected server, programs using DSL noticed alarms on the screen. As a result, in legacy server, the response time is between 12,478 and 13,021, and the average value is 12,750. For the aspected server, the response time is shown to be between 12,497 and 13,112, while the
average value is 12,805. Figure 8 is the result of the experiment above. In this way, the average overhead value of fault operational aspect is approximately 55 milliseconds. The lowest value, which is about 19 milliseconds, and the highest value which is around 91 milliseconds, are numbers that can be taken lightly. Therefore, the overhead of fault operational aspect resulted to be very low.

**Figure 8** The response time of fault operational aspect (see online version for colours)

- **The performance of configuration operational aspect:** This was conducted ten times, and the server was tested when turned on. It was tested when the connection method was called in the server. Legacy servers printed the completion message on the screen when the connection was complete. Aspect server created topology graphs on operational object. As a result, in legacy server, the response time is between 7,467 and 7,873, and the average value is 7,760. For the aspected server, the response time is shown to be between 7,673 and 8,037, while the average value is 7,855. Figure 9 is the result of the experiment above. In this way, the average overhead value of fault operational aspect is approximately 185 milliseconds. The lowest value is about 206 milliseconds, and the highest value is around 164 milliseconds. When the servers are first ran and connected with one another, for there is the cost of creating configuration graphs, the value is slightly high. However, for it is only executed when the server configuration is modified, this also has a negligible value.

- **The performance of accounting operational aspect:** The reading, updating, blocking, and releasing of the user was tested. 1,000,000 users went through testing. Four queries for each ten seconds during ten minutes were executed. Both legacy and aspected servers summed up query time were calculated. As a result, in legacy server, the response time is between 26,236 and 26,437, and the average value is 26,387. For the aspected server, the response time is shown to be between 26,286 and 26,529, while the average value is 26,318. Figure 10 is the result of the experiment above.
Thus, the average overhead value of fault operational aspect is about 185 milliseconds. The lowest value is about 206 milliseconds and highest value is around 164 milliseconds. For the response time resulted in 240 queries, this is also negligible.

4.2 Language expressiveness

Our language was evaluated by 15 professional game server operator users of the age of 27–35. The test subject groups were organised according to sex. They are consisted of 14 males and one female subject. For it was a day and night shift operation, there were
not as much women as there were men. We explained the purpose of this experiment for about 10 minutes. They were required to read the source code programmed by operational language which contained full fault, configuration, accounting, performance, and security features. The experiments ran for approximately 1 hour. They were conducted by language expressiveness: attractiveness, understanding and readability. For the quality of language expressiveness, we asked participants to complete a questionnaire rating their experience (using a 1- to 5-point scale, where 5 is the highest quality rating).

**Figure 11** Experimental result evaluated by operators (see online version for colours)

![Graph showing evaluation results for fault (F), configuration (C), accounting (A), performance (P), and security (S) with attractiveness, understanding, and readability dimensions.]

The results are shown in Figure 11. F, C, A, P and S stand for fault, configuration, accounting, performance and security. And the chart shows the evaluation results. As it is seen in the results, our language was attractive, but we received feedback regarding the moderation for accounting and security. It was discovered that structured query language (SQL) is more familiar to subject groups when handling accounting management using SQL. In addition, for the security management handled by commercial software, the language features of accounting and security management should be improved.

## 5 Conclusions

In this paper, we proposed DSL approach within FCAPS model. We proposed GOQL, as which operators can easily operate servers without programming expertise knowledge. Experimental result shows that GOQL was attractive to the operators. But language features of account and security managements need to be improved. This does not only have more than one operation going on at a time when resulting problems in revising initial bugs in applications, but it also has a negative influence that developers fail to apply the users’ needs instantly and continuously. As a solution to these problems, we present a language that operators or novice developers may utilise. By using the operational model, we designed a DSL. Our language contains facets on fault, configuration, accounting, performance, and security, and the operator can easily manage games. Our experimental results show that the fault, configuration, and performance design elements received high scores from users, and moderate scores on security and
accounting. From our experimental results, it became known to us through an interview that operators were more familiar with SQL, which seems helpful in accounting. Also, in security, commercial tools were used in the procedure which resulted in the moderate scores on this facet. For the future research of the language, there will be use of visual language. Rather than textual languages, visual languages are much more convenient and intuitive for operators or novice developers to handle. For this research, further study on visual notations, which are UX related elements, is needed. Additional studies on text-to-visual transformation used in automatically visualising text, syntax, and semantics is necessary as well.

References


Google Patents (2014) ‘Security auditing method based on aspect oriented programming (AOP) and annotation information system’.


