

A Multi-view API Impact Analysis for Open SPL Platform

Eunyoung Kim*, Kangtae Kim**, Hoh Peter In*

* Department of Computer Science and Engineering, Korea University
Anam-dong Seongbuk-gu, Seoul, 136-701- Korea

** Samsung Electronics

Maetan-dong 461 youngtong-gu, Suwon- Korea
marine0704@korea.ac.kr, kangtae.kim@samsung.com, hoh_in@korea.ac.kr

Abstract— Open platform is introduced and has become popular in the domain of consumer electronics. One of the main issues dominating the field of platform-based development concerns how to better maintain platforms to secure intended productivity and quality. A platform provides its service functions by means of APIs. Therefore, the key to successful development of a platform lies in successful development of relevant APIs. In this context, evolution of APIs, which accompanies that of platforms, is a factor that inevitably occurs and requires our constant attention.

This study proposes a multi-view impact analysis model for platform maintenance. This model traces APIs and their correspondent work products such as implementations, test cases and applications that use APIs. In addition, tools and ideas have been proposed as well to address the migrating nature of applications.

Keywords— API, Impact Analysis, platform evolution, open platform, Software Product Line

I. INTRODUCTION

Open platform is introduced and has become popular in consumer electronics domain[10]. Unlike the traditional approaches under which a company provides services based on the predefined needs of customers, the advent of open platform enables self-generation of the services in need. One of the key success factors in the ever-changing market is to have the ability to deliver, as soon as possible, the features that customers want. Using an open platform, a developer could add features or functions that the platform vendor had not completed or had not conceived of. An open platform allows a developer to amend the existing functionality.

In other words, with regard to development of the open platform, the control over applications has been handed over to an external developer or a 3rd party, other than the original platform developer. These characteristics make it harder to maintain APIs on an open platform. API changes can break the existing applications on the platform.

In the studies on the evolution of APIs, several former researches focused on how to update applications against API's changes[1][3]. Specifically, they focused on the traceability of API changes that differed from the previous version of a framework[4][5]. When distributing a new API on the framework, comparison was conducted with the

existing API to detect the API changes. This detection mechanism, however, becomes available only after changes are made to the framework.

From the viewpoint of platform maintenance, it is important to secure the traceability of applications, when a change is made to the API. No less important is how to efficiently maintain the API itself upon occurrence of additions and changes arising out of the need for new functions. It is necessary to analyze, prior to the changes, the possible influence of the proposed changes. So, platform maintainer can determine whether or not the changes can bring about the benefits worthy of the efforts[6].

Therefore, a novel approach to API impact analysis is in need to better maintain open SPL platforms. The approach should take into account the following factors:

First, traceability should be guaranteed of the applications whose development has used APIs of the platform. It should be determined whether the previous application has migrated or not, once a new version is released. Successful tracing of API-based applications leads to successful detection of the areas affected by the changes. The information further leads to determination of whether it is beneficial to modify the existing applications or to develop new ones.

Second, it is also necessary to be able to trace the internal components of platform that are affected by API changes. The need arises out of the necessity to maintain the platform in such a way as to provide new features required of the platform at minimum costs. For example, consider conducting an impact analysis of a change that is made to API. The change is traced to all the classes that implement the API and the derived APIs. Those classes may require some modifications as well. Modified classes should be subject to test to assure that the changed functions work without any side effects. Based on the analytical results, tradeoffs can be performed between two decisions: whether to create a new API in consideration of the affected area, or whether to modify the relevant outputs, while maintaining the existing API.

Proposed herein is a method of analyzing multi-view API impacts on open SPL platforms in the embedded software. It analyzes impacts of API changes with traceability between APIs and their correspondent work products such as API implementations, TestCase and applications. It is applied to open platforms for the B2B market in the printing domain. The

platform provides customization for components by supporting Java and other Web technologies. We adopted the API impact analysis method to three releases of the platform. From the source code of each of them, we were able to trace the applications that used the APIs, the implementation classes that realized the APIs and constitutes the internal structure of the platform, and the TestCases that tested them.

In the remainder of this paper is organized, as follows: Section 2 describes the literature on open platform and on analysis of impacts caused by changes; Section 3 sets forth the overall review of our proposed API analysis and the details about what is suitable for open platform of the printing domain; Section 4 introduces a case study by applying our model to three releases; and, Section 5 concludes our study with suggestions for future works.

II. RELATED WORKS

A. Open platform trend

The consumer electronics industry is shifting its attention to open platform-based software development. Let's take the mobile sector for example. The competition in the mobile platform market is getting fiercer, as the spectrum of the market participants gets wider and wider. For instance, tapping into the market are not only the traditional OS vendors such as Microsoft and Symbian, but also search engines such as Google and Yahoo! and manufacturers like Nokia and Samsung. RTOS-based feature phones are losing grounds in the market, while open platform-based smart phones are gaining a speedy momentum. As a result, the mobile industry is faced with a higher demand for more services and functions. To catch up with the demand, APIs have been evolving. Previously, it was sufficient for an API to be able to provide limited functions tailored for local resources. Now, however, it is being transformed to help create a variety of services through disclosure of APIs to the public through open platform.

An open platform should be able to help a third-party developer implement functions currently not available on it, and reimplement or replace the existing functions[7]. Securing the openness necessitates constructing a reconfigurable architecture that allows interchangeability among components, and defining an API to better use the architecture. To define the API for that purpose, it is, in turn, necessary to part away from the traditional designing. In other words, focus is not on functions of a single application; rather, it is on the issue of how to design, in consideration of common functions among multiple applications, the architecture to prevent duplication and to accommodate the functions that might become necessary in the future [8].

III. A MULTI-VIEW IMPACT ANALYSIS MODEL

This section describes the API impact analysis views proposed herein. The views consist of two categories: One is about analysis of applications using the API of a platform to support migration of applications; and, the other is about tracing platform internal work products to support evolution of the platform.

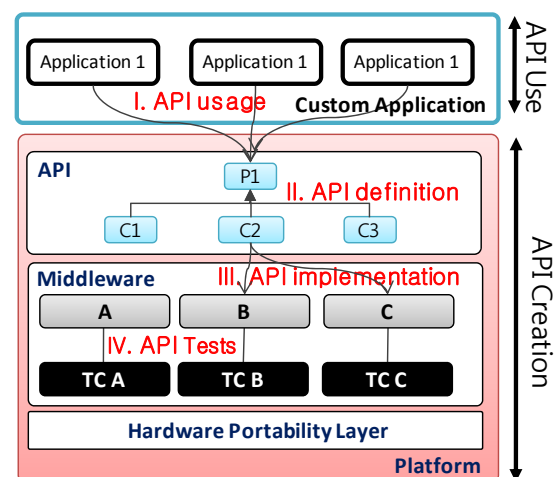


Figure 1. A Multi-view API Impact Analysis Model

Figure 1 illustrates the proposed API impact analysis model. This model consists of two categories related to generation and use of APIs, and four sub-views for analysis of them. Line I indicates the process of analyzing how an application uses the API of the platform; namely, Line I analyses the API use category. Lines II to IV depict the internal structure related to provision of platform functions to the API. Specifically, Line II analyzes the inheritance process, while Line III analyzes the classes that maintain implementing relations with the API. Line IV traces the TestCase that is to verify whether or not the API function has been implemented as intended.

In the next sub-section, the impact analysis views are to be explained in detail, which are shown in Figure 1.

A. Application View

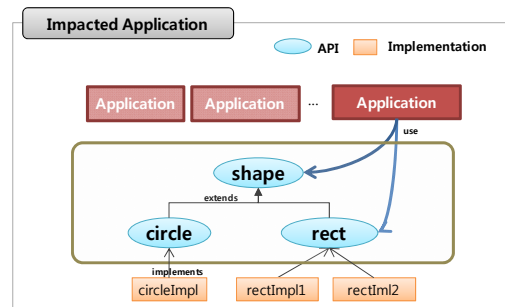


Figure 2. API-based Application

This view traces applications that use the APIs of the platform. Applications are not executable on the platform, once its method provided by the platform is removed or the parameters thereto are changed. As shown in Figure 2, for example, if a change occurs to a parameter, the application that calls it should be changed. More specifically, if the draw (e.g. int p1, int p2, int p3, and int p4), which represents the method of the rect API, is changed to another draw (e.g. int start, int width, and int height), the calling method shall be altered as well.

An application can use the rect API in either of the two ways; namely, generating, for reference, an instance of the

rectImpl1 for reference, which is an API-implementing class, or using the rectImpl1 as reference variable of rect or a shape type.

The following table sums up how an application uses an API:

Table 1. API Usage Types for Application

Usage Type	Sample Code
Variable Reference : API type	rect r = new rectImpl1();
Variable Reference: Parent API type	r = new rectImpl();
Parameter Type	void DoubleDraw(shape s)

• Affected application search algorithm

Input: one of APIs A

apps = getApplicationUsage(A);

while (a $\neq \emptyset$) **do**

(a) = getParent(A);

apps = apps U getApplicationUsage(a);

replace A with a

end

The algorithm uses the following functions: (i) The getApplicationUsage(A) returns a list of classes that generate instances of A type, or contains codes used as parameters of a method; or (ii) the getParent(A) returns a parent API of A.

B. API View

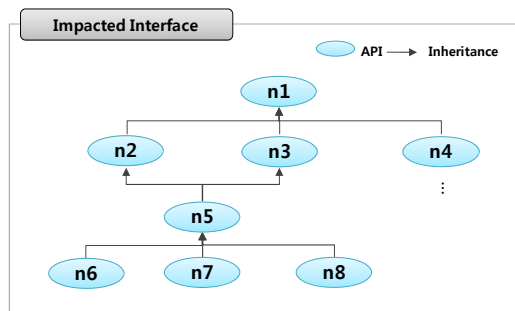


Figure 3. API Inheritance Tree

An architect defines APIs by means of inheritance of extensibility. Here, inheritance implies that a class can reuse the data members and the member functions of its parent class. Therefore, the parent class and the child class form a mutual dependency, and end up affecting each other. This view traces both of the APIs to be modified and those belonging to the same inheritance tree. Therefore, it becomes possible to identify all the APIs that are indirectly affected by the APIs to be changed. If the n2 in Figure 3 is changed, the change influences its implementation, and the applications using it. Therefore, ripple effects affect how the children APIs inheriting n2 (i.e. n5, n6, n7 and n8) are implemented, and how applications use them. On the other hand, the n6 affects the directly affected implementation classes and the relevant applications alone.

• Affected API search algorithm

Input: A set of APIs A

while (A $\neq \emptyset$) **do**

(a) = nextAPI(A);

childs = children(a);

for each c $\in$ childs **do**

if (hasChildren(c))

childs= childs U children(c);

end

Remove a from A

end

The algorithm use the following functions: 1) nextAPI(A) returns the first item in the chin A, 2) children(a) returns a list of classes that inheri from a, and 3) hasChildren(c) return true if number of children(NOC) of c > 0 , false

C. Implementation View

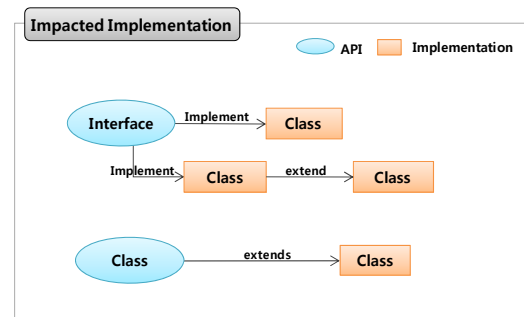


Figure 4. API Implementation Relationships

Analysis is carried out of the relationships that are involved in implement APIs. This view is defined based on Java. Java supports use of an interface or a class as an API.

Therefore, the relationship of the affected implementation, which is subject to analysis, varies depending on API types. Figure 4 shows two ways of implementing an interface. One is to generate a class that implements the body of an interface's abstract method. Herein, the keyword is "implements." The other method is to inherit and extend the previously implemented class. In the latter case, "extends" is the keyword to use. Just as in the case of a general class, it is possible to implement a class defined as an API through inheritance.

Analysis of API implementations requires that of the following three cases:

Table 2. Types of API Implementation

API Type	Sample Code
Interface	Class rectImpl implements rect
Interface	Class roundRectImpl extends rectImpl
Class	Class rectException extends Excpction

- Affected implementation search algorithm

Input: A set of APIs A

```

while( A ≠ ∅ ) do
  ( a ) = nextAPI(A);

  if ( a.type = "Interface" )
    impls = getImplements( a );

    for each i` ∈ impls do
      impls = impls U getExtends( i` );
    end
  end

  else if ( a.type = "class" )
    impls = getExtends( a )
  end
  Remove a from A
end

```

D. TestCase View

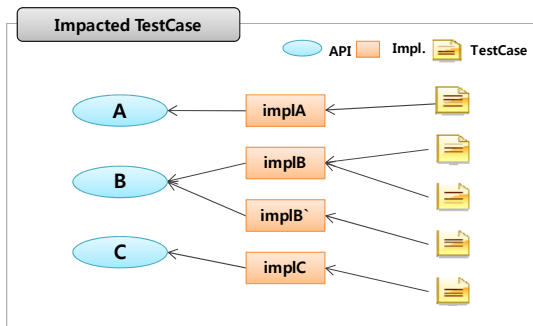


Figure 5. Relationship between API and TestCase

This view traces the TestCases that are mapped onto APIs. Analysis of affected TestCases begins by identifying the implementation class of the API to which the change is made. Subject next to tracing are the TestCases that test implementation classes. Figure 5 shows the relationship between APIs and TestCases. A single API can have multiple implementation classes that are tested by several TestCases. For example, the API B has the implB and implB`. The implB has multiple TestCases, because a TestCase is created as a unit of an API method. The TestCase view requires, as its input, the previously analyzed implementation view and the TestCase data. First, the implementation class that implements the corresponding API should be retrieved, based on the name of the API, from the implementation view. Then, the names of APIs and methods are mapped onto the TestCase data. A tester prepares the TestCase data, which consist, among others, of test scenario, test procedure, input, and output.

- Affected TestCase search algorithm

Input: A set of APIs A

```

while( A ≠ ∅ ) do
  ( a ) = nextAPI(A);

```

```

  Impls = mappingImplement( a );

```

```

  for each i` ∈ impls do
    TCs = mappingTestCase( a );
  end
  Remove a from A
end

```

The algorithm use the following functions: 1) mappingImplement(a) returns a list of implementation classes that are matched with a name with API name in implementation view. 2) mappingTestCase(a) returns a list of TestCases that are matched with a name with API name in TestCase data.

IV. A CASE STUDY

This section introduces a case study wherein our proposed API impact analysis model has been applied to P platform in the printing domain. P platform is an open architectural platform for the B2B market. It supports customization and extensibility of various functions of the next generation multi-function printers. Any developer and independent software vendor may develop his or her own application using the APIs of P platform.

Table 3. Sizes of Released Codes for P Platform

Release	A	B	C
SLOC	1,121,104	1,203,430	1,909,458
Class #	4,821	7,014	8,732
API #	1,667	1,691	1,972
Interface	264	263	320
Abstract Class	225	235	251
Class	1174	1212	1401

To verify the validity of our proposed API analysis model, analytical tools have been developed, based on the model. The tools extract the necessary information for each view through the static analysis of source codes. Table 3 outlines the analytical results on each release of the platform. The numbers shown in the table were counted upon the release of each version. The category “class number” indicates the number of the classes defined for implementation of APIs, while the category “API” refers to the number of the interfaces and classes that were disclosed to the public. APIs were classified into the following three classes: interface, abstract class and class. Especially, APIs such as data type, enumeration and exception handler are termed as “class,” since they do not need any type of implementation.

In this section, the analytical results for each view are explained from either of the following two viewpoints: (Due to spatial limitations and security issues, the results concerning C releases are to be explained alone.)

A. Application Analysis

As explained in the preceding section, it is possible to use applications by creating not only relevant APIs, but also parent APIs. Therefore, the applications affected by the Queue API

include both of the applications using the Queue and those using the parent API. Table 4 lists some of the applications using Queue APIs. It shows that the method of passing Queue API-type parameters is located in the BoxContainer class of the Box plug-in.

Table 4. Queue API-based Applications

Layer	Application	Class	Usage Type
Plug-in	Box	BoxContainer	Parameter
Plug-in	Scan	JobHandler	Variable
Plug-in	Scan	ScanManager	Parameter
...			

Table 5 only shows how the Queue and parent APIs are used. Parent 1 was used three times as a variable, while Parent 2 was used as a parameter four times. Thus, the number of applications affected by the changes to the Queue API represents the sum of the number of the applications using the Queue and that of the applications using parent APIs.

Table 5. Usage of Queue API with Applications

	Parameter #	Variable #	Total
Queue	3	4	7
Parent 1	0	3	3
Parent 2	4	0	4
Parent 3	1	2	3
...			

B. Platform Analysis

Two cases were cited to explain the impacts of the API changes on the platform. One was the API at the lowest leaf node of the API structure, and the other was the API that has children APIs. Let's take Figure 2 for example. Concrete APIs like the rect and the circle represent the former, while abstract APIs like the shape represent the latter. The existing platform was free from any impact, even upon addition of concrete APIs like triangle.

Thus, to use applications in need, one had to simply add triangle APIs inheriting shape APIs, and to implement them. However, when a change was made to the rectangle API, modification should be made as well to the class that implements it, and such modification should be subject to a test to verify its operability. In addition, when a change was made to the shape API, the affected area got enlarged, because it was necessary to implement newly added or modified methods of the circles and the rectangles inheriting the shape. Table 6 charts out the analytical results of two APIs on either end of inheritance. The concrete API has a single implementation class, and 29 TestCases. On the other hand, in the case of the abstract API, it is necessary to trace and analyze not only the affected APIs, but the children APIs.

Table 6. Usage of Sample APIs on Platform

	Children API #	Impl. Class #	TestCase #
Concrete API	0	1	29
Abstract API	41	28	-
Child 1	20	3	-
Child 2	11	0	-
Child 3	0	1	28
...			

C. Comparison of Analytical Results of Two Versions

By compare with two releases, it is possible to identify changed API from previous version and to predict the effort for modification of affected implementation classes and TestCase by the change. Figure 6 illustrates the percentage share of the implementation classes of the C release that has been affected by the API changes, juxtaposed with the composition of the B release. When compared, 74 APIs were changed, along with 161 classes that implemented them. In detail, the left pie graph shows the share of the implemented APIs out of the total 571 APIs that were defined by the interfaces and the abstract classes requiring implementation in the C release. Of the 571, 446 APIs, or 78% of them, had implementation classes. Specifically, only 74 of the 446 APIs had changed methods, while the 372 APIs of them remained unchanged. A single API had multiple implementations. Therefore, the aggregate number of APIs was outnumbered by that of their classes. In the meanwhile, the right pie represents the implementation classes of the C release. To implement 446 APIs, 3,729 classes were mobilized. 4% of the total 3,729 classes, or 161 classes, needed modification upon change to the APIs.

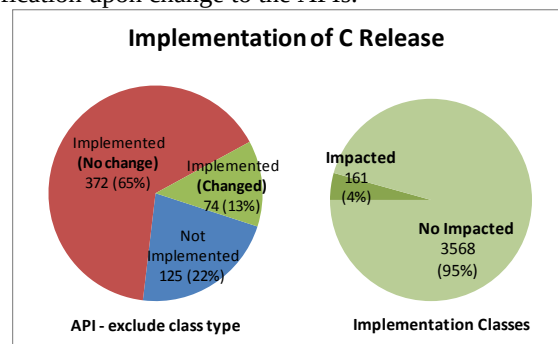


Figure 6. Affected Implementation Upon Comparison of B with C

Figure 7 shows the portion of the TestCases of the C release affected by the API changes, through comparison of that of the B release. A total of 809 TestCases were used to conduct a series of tests on the 42 APIs of them, and 195 cases were affected. Based on the results, it seems that the graphs indicate only 3% of the cases (i.e. 42 cases) have been subject to test, because the information does not incorporate the data on the previously conducted TestCases. Combinations of APIs' and their methods' names served as keys in mapping of the APIs and their TestCases. Modification is ongoing, since the previous TestCases names differ from the new ones.

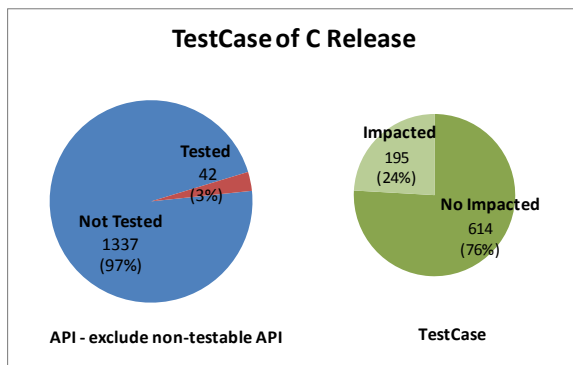


Figure 7. Affected TestCase by comparing with B and C release

V. CONCLUSIONS

In this study, a model for analysis of multi-view API impacts has been proposed, which is designed to better maintain APIs of the B2B open platform used in the printing domain. Our proposed model analyzes the areas that are affected in the course of implementing and using the APIs of the open platform. The model has been applied three times to the releases of an open platform currently under development. As a result, it was possible to trace classes and applications affected by the API changes.

Prior to the present study, developers did not know exactly how much APIs were affected, and how much implementation classes of a platform were to be modified to cope with the impact. Upon receipt of new API specifications, the developers determined, on their own, whether or not any changes had been made to the existing APIs, and conducted modification if necessary. In short, developers repeated a boring cycle of modifying and building, until no error was found out. In addition, testers, who lacked the background information on the platforms, found it extremely difficult to detect API changes, only based on specifications, and, consequentially, lost a considerable portion of the crisscrossing impacts among APIs.

Our model is believed to have contributed to resolving the inconveniences by providing better information to the stakeholders involved in development of a platform.

First, objective data are provided to stakeholders by analyzing all the work products related to APIs.

Second, since the API impact analysis is based on source codes, it is possible to carry out the updated analysis at any time.

Finally, analytical tools have been developed as well to dramatically reduce the time consumed for analysis. In the past, it took about a week for a tester to determine, based on the recorded specifications, whether or not changes had been made to APIs, and to trace relevant TestCases. Development of the tools has led to production of improved quality just for a one-day period.

More efforts will ensue in the future to improve the traditional models as well as the proposed one; and, thereby, to expand the scope of the impact analysis to the level of method or data and to improve the analytical accuracy. The

proposed model takes classes, which constitute APIs, as its unit. Thus, it might reveal inconsistency concerning the areas that do not affect actual changes. When analyzing an application, no method is affected as long as the changed method is put to use. In addition, even when creating and using an instance of an API, the method also remains unchanged. The proposed model, however, fails to detect the difference. In this context, it is necessary to narrow down the analytical unit from the current API class to the API method.

We also plan to develop tools that are helpful for the independent developers who develop applications as part of the SDK of P platform. It is vital for an application developer to be accurately aware of the platform version through which relevant applications can be executed. Therefore, it is necessary to develop tools that check the platform version that is compatible with an application, and, to trace changed APIs and modify them.

ACKNOWLEDGMENT

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2009-(C1090-0903-0004))

This research was supported by National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development and Experts Education.

REFERENCES

- [1] I. Savga, M. Rudolf, J. Sliwinski, J. Lehmann, H. Wendel, *API Changes - How Far Would You Go?*, 11th European Conference on Software Maintenance and Reengineering (CSMR'07), 2007
- [2] Dig, D., Johnson, R. *How do APIs evolve? A story of refactoring*, Journal of Software Maintenance and Evolution 18, 2006 Page(s) 87–103
- [3] B Gharaibeh, TN Nguyen, JM Chang, "Coping with API Evolution for Running, Mission-Critical Applications Using Virtual Execution Environment", 7th International Conference on Quality Software (QSIC '07), 2007
- [4] B Dagenais, MP Robillard, "SemDiff: Analysis and Recommendation Support for API Evolution", 31th International Conference on Software Engineering (ICSE '09), 2009
- [5] J.Henkel, A. Diwan. "CatchUP!: Capturing and replaying refactorings to support API evolution", 27th International Conference on Software Engineering (ICSE'05), 2005
- [6] McGregor, John D. "The Evolution of Product Line Assets (CMU/SEI-2003-TR-005, ADA418409)". Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr005.html>.
- [7] Wikipedia "Open platform". En.wikipedia.org/wiki/Open_Platform, Oct. 2009.
- [8] Wills, L.; Kannan, S.; Sander, S.; Guler, M.; Heck, B.; Prasad, J.V.R.; Schrage, D.; Vachtsevanos, G., "An open platform for reconfigurable control"; Control Systems Magazine, IEEE Volume 21, Issue 3, June 2001 Page(s):49 - 64
- [9] Mulligan, C. "Open API standardisation for the NGN platform, Innovations in NGN: Future Network and Services", 2008. K-INGN 2008. First ITU-T Kaleidoscope Academic Conference, 12-13 May 2008 Page(s):25 - 32
- [10] Nordström, B., *Ericsson Business Review* 2, 2007 pp58-60