UDC 004.052

**A.S. GONCHAROV, A.A. FURMANOV**

***National aerospace university named after N.E. Zhukovsky «KHAI», Ukraine***

# MULTI VERSION SERVICE ORIENTED ARCHITECHTURES DEPENDABILITY ANALISYS

*This article is targeted on investigating the web-services dependability problem and, in particular, the reasonableness of multi version architecture usage to solve it and guarantee the customer he is safe while using the web-service. It provides a reader with general information about web-services architecture, challenges that such architecture faces and ways of solving such problems. The design of the model used for analysis of multi version architecture is represented. Results of comparison of different model configurations are carried out showing the vulnerabilities of each configuration. The efficiency of multi version service oriented architecture is proved on a basis of the experimental research.*

***Key words:*** *web-service, Service Oriented Architecture, SOA, dependability, vulnerability, intruders, multi-version, majority element, web server, application server, database management system (DBMS).*

## Introduction

Companies have long sought to integrate existing systems in order to implement information technology (IT) support for business processes that cover all present and prospective systems requirements needed to run the business end-to-end. A variety of designs serve this end, ranging from rigid point-to-point electronic data interchange (EDI) interactions to web auctions. By updating older technologies, for example by Internet-enabling EDI-based systems, companies can make their IT systems available to internal or external customers; but the resulting systems have not proven flexible enough to meet business demands that require a flexible, standardized architecture to better support the connection of various applications and the sharing of data.

Service Oriented Architecture (SOA) [1] provides a design framework with a view to realizing rapid and low-cost system development and to improving total system-quality. SOA offers one such prospective architecture. It unifies business processes by structuring large applications as a collection of smaller modules called "services". Different groups of people both inside and outside an organization can use these applications, and new applications built from a mix of services from the global pool exhibit greater flexibility and uniformity. One should not, for example, have to provide redundantly the same personal information to open an online checking, savings or IRA account, and further, the interfaces one interacts with should have the same look and feel and use the same level and type of input-data validation. Building all applications from the same pool of services makes achieving this goal much easier and more deployable to affiliated companies. For example: interacting with a rental-car company's reservation system from an airline's reservation system.

**Web services face significant challenges** because of particular requirements. Applying the SOA paradigm to a real-time system presents many problems, including response time, support of event-driven, asynchronous parallel applications, complicated human interface support, reliability, etc.

**The other challenge** is that almost all of business task that should be integrated into services must be as dependable as possible cause the end user should feel himself the same as if he is speaking with the company consultant or manager tete-a-tete. The problem is that web-services as any other software solutions can provide unexpected result while processing customer’s request because of any hardware failure, software failure or hacking attempt. If hardware failures can be found nowadays quite rarely due to the fact that modern hardware architecture is being known quite well, the problem of an attack is growing day by day and the accent is now targeted to breaking business critical applications to get real profit on it, cause idle stand of business-critical application for just one minute can turn into millions of dollars loss.

The well known methods of protection from network attacks are firewalls and hardware systems of attack detection, but nowadays much time is spent on developing the software architecture that could allow increasing web-service dependability even more. This is a multi version architecture based on usage of several operating channels with different software and hardware configuration. This solution should reduce the weight of the platform issues in a scope of failure reasons.

In this artivle I will try to prove the resounabless of such

an architecture usage - actually I will try to check the accuracy of results get earlier. Most works on this problem I have found while analyzing this branch [2] provide just simple models of such architecture that can be used only once for the investigation, cannot be applied to some real environment and even does not use actual web-service interaction which, as for me, can distort the result of the actual experiment environment that should give more reliable results. The model used in the experiment interacts with a real web service.

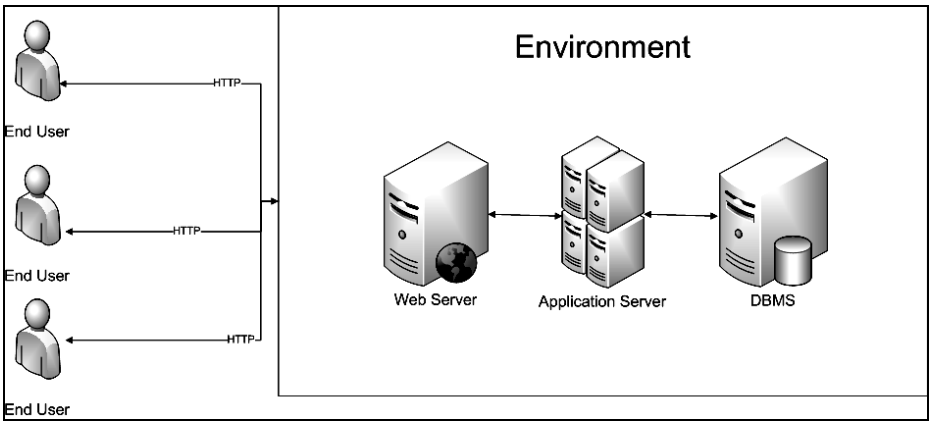


Fig. 1. Typical environment for SOA and its simplest way of interaction with end users

## 1. Dependable Service Oriented Architecture analysis

Common service provider consists of three parts: Web Server, Application Server and Database Management System (DBMS), running over Operating System and Hardware.

Fig. 1 represents the typical environment used for SOA [1] that interacts directly with end users. Such kind of interaction is not used in practice but it can give us an understanding of what's happening in service. So end user sends a request to web-service using http (or https) protocol and is waiting for response. On the server side web server receives the response, that it redirects the response to the application server which interacts with the DBMS if needed and gives a result to web server whose work now is just to translate the result to the end user. In practice firewalls and other protection elements are placed between the end user and environment. But from this picture we can clearly see that failures can take place on such levels: environment hardware failure, environment software failure, http transfer failure and effect of possible intruder between end user and environment. In this article we'll focus on the intruder affects the environment so that it leads to a failure as a result. The most frequent way for the intruder to affect the environment is to perform a request pointed to software vulnerabilities on a target environment. Such request is called an attack.

The proposed way of solving such a problem is providing architecture more stable to attacks making the intruders work harder by applying majority backup for the request channels using the method shown on fig. 2 [3, 4].

The advantages of such architecture are:

– the tolerance to network attacks is much greater, because it is much harder for intruder to rise a fault on every environment;

– availability is increased;

– performance is also increased due to hot restore scheme usage.

But as any other solution it has some disadvantages:

– hardware complexity and high prices;

– software complexity;

– security problems because of just one environment hacking causes data loss.

## 2. Muli version architecture efficiency

### 2.1. Modeling system used in research

The design of the model used is represented on fig. 3. The Generators part has access to the vulnarabilities database.

The database used for the research is National Vunarabilities Database (NVD) [5, 6]. TaskExecutor is a web servis run on Apache web server under the AXIS 2 environment. The MajorityElement module emulates the behavior of the multi version SOA using the "2 of 3" criteria.

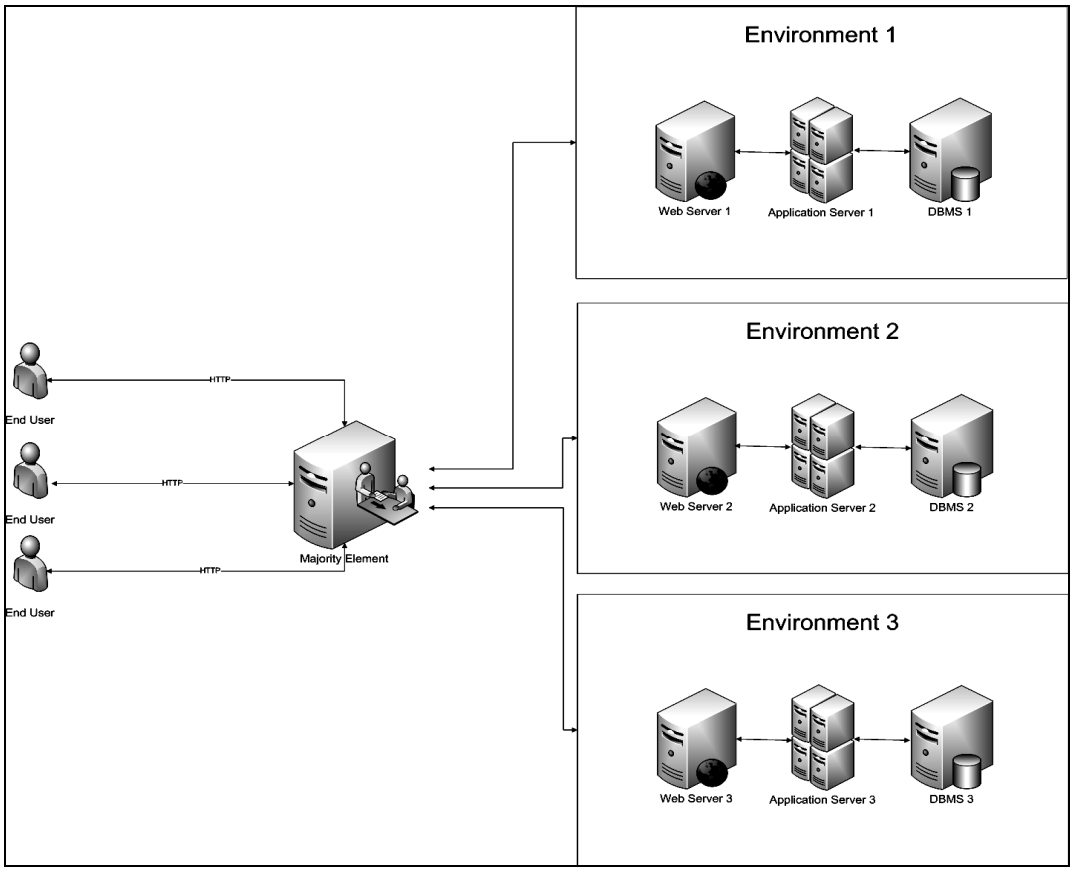


Fig. 2. Web-service building architecture with Majority Element usage

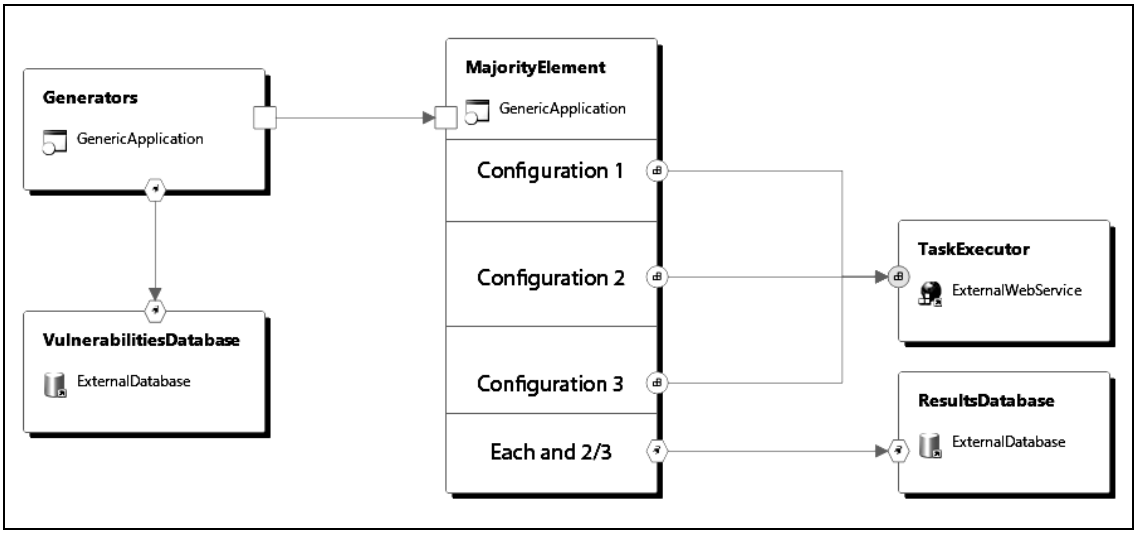


Fig. 3. Modeling system base architecture design

Each emulated configuration (component base – CB), as described in a first stage, contains the following elements: Web Server (WS), Application Server (AS) and Database Management System (DBMS):

$$CB = \{WS; AS; DBMS\}. \quad (1)$$

Each service consists of the components of all types, so the different sets of components are available:

$$WS = \{WS_i\}_{i=1}^{n_{WS}}, \quad (2)$$

$$AS = \{AS_i\}_{i=1}^{n_{AS}}, \quad (3)$$

$$DBMS = \{DBMS_i\}_{i=1}^{n_{DBMS}}. \quad (4)$$

Each component contains the vulnerabilities set:

$$\forall WS_i \approx VS_{WS_i} = \{V_{WS_ik}\}_{k=1}^{VS_{WS_i}}. \quad (5)$$

Each attack is defined with a set of vulnerabilities and repetition factor:

$$A = \left\{ A_j = \left\{ R_j, \Delta V_j \subset V \right\} \right\}. \quad (6)$$

The attack influence on web-service is shown on fig. 4.

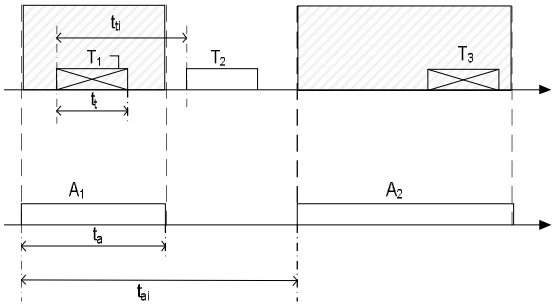


Fig. 4. Attack influence on service:
$t_a$ – attack time, $t_{ai}$ – interval between attacks;
$t_t$ – task execution time, $t_{ti}$ – interval between requests;
$A_i$ – attack; $T_i$ – executed task.

The component base configurations used during the experiment are shown in table 1.

Table 1

Environment configurations

| # | **Web Server** | **Application Server** | **DBMS** |
|---|---|---|---|
| 1 | Apache | Apache Tomcat | MySQL |
| 2 | IIS | BEA Web Logic | PostgreSQL |
| 3 | thttpd | IBM Web Sphere | Oracle8i Database Server |

### 2.2. Model configurations analysis

After analyzing the NVD [5] the table 2 is built showing the vulnerabilities of each configuration. From the table we can see that the most dependable configuration is Configuration 3.

Table 2

Investigated configurations vulnerabilities

| **Configuration** | **Components** | **Vulnerabilities** | **Configuration vulnerabilities** |
|---|---|---|---|
| Configuration 1 | Apache | 17 | 54 |
| | Apache Tomcat | 5 | |
| | MySQL | 32 | |
| Configuration 2 | IIS | 6 | 83 |
| | BEA WebLogic | 61 | |
| | PostgreSQL | 14 | |
| Configuration 3 | thttpd | 4 | 44 |
| | IBM WebSphere | 30 | |
| | Oracle8i | 10 | |

It is supposed that the experiment will lead to the same conclusion. It is also clear that the more complex software component is – the more vulnerabilities are found in it.

### 2.3 Experiment order

To run the experiment the modeling software has been run in different configuration of such parameters as:

- vulnerability replication factor;
- task execution time;
- request generation interval;
- attack effect time;
- attack generation interval.

Because of the experiment was hold mainly to compare the results with the modeled service version [2] the uniform distribution of parameters is used. In each experiment the dependbility coefficient was calculated for each configuration and for majority element regarding the formula:

$$C_{dep.} = \frac{Q_{succeded\_tasks}}{Q_{tasks}}, \quad (7)$$

where $C_{dep}$ – a dependability coefficient;
$Q_{succeded\_tasks}$ – a quantity of succeded tasks and;
$Q_{tasks}$ – a quantity of tasks generated.

### 2.3. Experiment results

Relationship between

- dependability coefficient and vulnerability replication factor is displayed on fig. 5;
- dependability coefficient and task execution time represents fig. 6;
- dependability coefficient and request generation interval is displayed on fig. 7;
- dependability coefficient and attack effect time shows fig. 8;
- dependability coefficient and attack generation interval is displayed on fig. 9.

## Conclusion

The experiment shows that the majority element usage is reasonable in cases when the attack effect time is short and attack generation interval is bigger than request generation one. The vulnerability replication factor also affects the majority element in a worse efficiency. Majority element is extremely efficient in the environments with short attack effect. The dependability coefficient of the service itself depends on all the factor token part in the experiment.

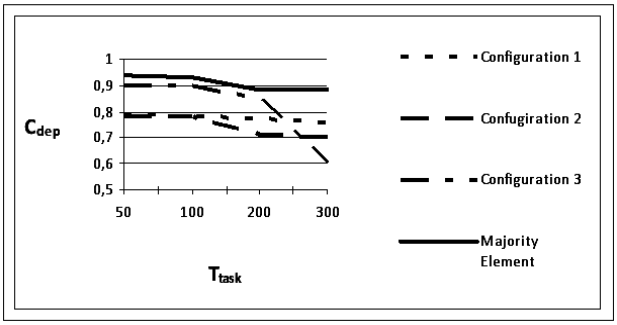


Fig. 5. Relationship between dependability coefficient and vulnerability replication factor

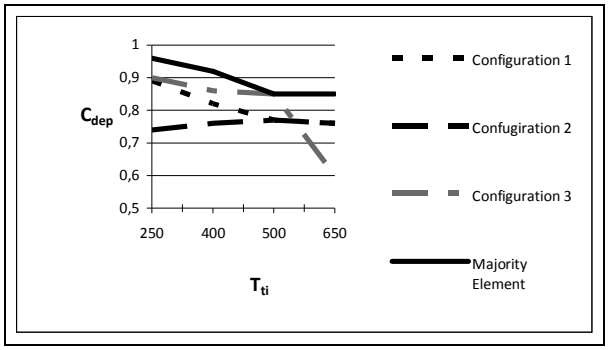


Fig. 6. Relationship between dependability coefficient and task execution time

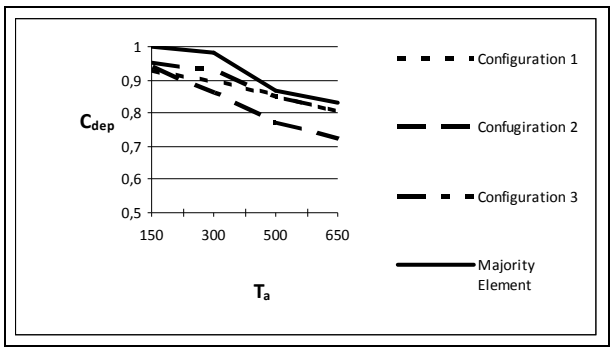


Fig. 7. Relationship between dependability coefficient and request generation interval

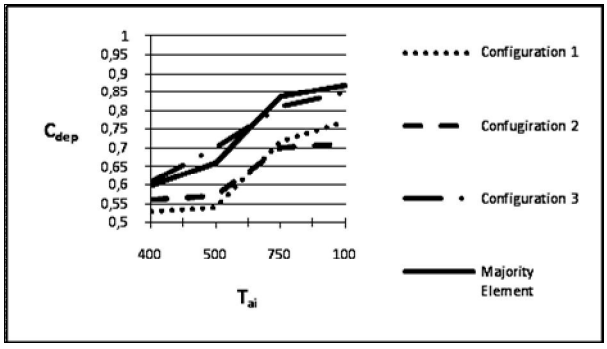


Fig. 8. Relationship between dependability coefficient and attack effect time

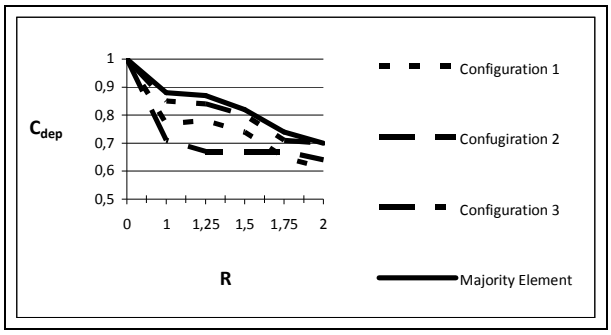


Fig. 9. Relationship between dependability coefficient and attack generation interval

The results got during current experimental research are quite close to the results retrieved earlier [2], but the behavior of diagram curves is not so slight as in theoretical researches so it makes me to believe that real web-service environment usage made some corrections to the results.

In future work it is reasonable to improve the in vestigation including the research of different configuration parameters parallel changing interaction, using other distribution of input parameters and extending the model to make it multi leveled request-response system.

## References

*1. Web Services Architecture [Электрон. ресурс]. – Режим доступа к ресурсу: http://www.w3.org/TR/ws-arch/.*

*2. Фурманов А.А. Моделирование гарантоспособных сервис-ориентированных архитектур при атаках с использованием уязвимостей / А.А. Фурманов, И.Н. Лахижа, В.С. Харченко // Радіоелектронні і комп'ютерні системи. – №7 (41). – Х.: ХАИ, 2009. – С. 65-69.*

*3. Secure, Reliable, Transacted Web Services: Architecture and Composition [Электрон. ресурс] / D.F. Ferguson, T. Storey, B. Lovering, J. Shewchuk // Microsoft and IBM Technical Report, Database. – 2003. – Режим доступа к ресурсу: http://www-106.ibm.com/developerworks/webservices/library/ws-securtrans.*

*4. Dependability in the Web Service Architecture / F. Tartanoglu, V. Issarny, A. Romanovsky, N. Levy. – In: Architecting Dependable Systems. – Springer-Verlag, 2003. – P. 89-108.*

*5. Furmanov A. A. The analysis of vulnerability databases for selecting dependable service-oriented architectures / A. Furmanov // Радіоелектронні і комп'ютерні системи. – Х.: НАКУ «ХАІ». – 2007. – № 8 (27). – С. 15-19.*

*6. National Vulnerability Database [Электрон. ресурс]. – Режим доступа к ресурсу: http://nvd.nist.gov/.*

*Поступила в редакцию 10.03.2010*

**Рецензент:** д-р техн. наук, проф., зав. кафедры компьютерных систем и сетей В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков, Украина.

## АНАЛІЗ ГАРАНТОЗДАТНОСТІ БАГАТОВЕРСІЙНИХ СЕРВІС-ОРЕЄНТОВАНИХ АРХІТЕКТУР

***О.С. Гончаров, О.А. Фурманов***

Розглядається проблема гарантоздатності веб-сервісов, зокрема доцільність використовування багатоверсійної архітектурі для її рішення, тим самим забезпечення безпеки клієнтам під час використання веб-сервіса. Читачеві надається загальна інформація що до архітектури веб-сервісів, викликів, що їх стикається така архітектура, та шляхи вирішення цих проблем. Описано проектування моделі, використовуваної для аналізу багатоверсійной архітектури. Проіллюстровані результати порівняння різних конфігурацій моделі, показані вразливості кожної конфігурації. Доводиться ефективність багатоверсійної сервіс-орієнтованої архітектири на базі результатів експериментального дослідження.

**Ключові слова:** веб-сервіс, сервіс-орієнтована архітектура, SOA, гарантоздатність, вразливість, зломщик, багатоверсійність, мажоритарний елемент, веб сервер, сервер пристосунків, система управління базами данних.

## АНАЛИЗ ГАРАНТОСПОСОБНОСТИ ДИВЕРСНЫХ СЕРВИС-ОРИЕНТИРОВАННЫХ АРХИТЕКТУР

***А.С. Гончаров, А.А. Фурманов***

Рассматривается проблема гарантоспособности веб-сервисов, в частности целесообразность использования диверсной архитектуры для её решения, тем самым обеспечения безопасности клиентам при работе с веб-сервисом. Читателю предоставляется общая информация об архитектуре веб-сервисов, вызовах, с которыми сталкивается такая архитектура, и пути решения этих проблем. Описано проектирование модели, используемой для анализа диверсной архитектуры. Проиллюстрированы результаты сравнения различных конфигураций модели, показаны уязвимости каждой конфигурации. Доказывается эффективность диверсной сервис-ориетированной архитектуры на базе результатов эксперимательного исследования.

**Ключевые слова:** веб-сервис, сервис-ориентированная архитектура, SOA, гарантоспособность, уязвимость, взломщик, диверсный, мажоритарный элемент, веб-сервис, сервер приложений, система управления базами данных.

**Гончаров Алексей Сергеевич** – студент факультета радиотехнических систем летательных аппаратов Национального аэрокосмического университета им. Н. Е. Жуковского «ХАИ», Харьков, Украина, e-mail: alexei.goncharov@gmail.com.

**Фурманов Алексей Аркадиевич** – ассистент кафедры компьютерных систем и сетей Национального аэрокосмического университета им. Н. Е. Жуковского «ХАИ», Харьков, Украина.