UDC 621.385

PATTERNS FOR RELIABLE WEB-SERVICES

AMER TAHSEEN SALAMEH ABU-JASSAR
V.N. Karazin Kharkiv national university

O.B. TKACHOVA
Kharkov national university of radioelectronics

В статті запропоновано підхід забезпечення надійності Web-сервісів в SOA, що базується на використанні спеціальних шаблонів проектування. Запропоновані шаблони дозволяють забезпечити як надійність доставки повідомлень, так і оцінити досягнутий рівень надійності сервісів.

- -

This paper is devoted to developing an approach to ensure the reliability of Web-services in SOA which based on the use of special design patterns. Proposed patterns provide reliable delivery of messages and patterns that allow estimate the current level of reliability.

- -

В статье предложен подход обеспечения надежности Web-сервисов в SOA, основанный на использовании специальных шаблонов проектирования. Предложенные шаблоны позволяют обеспечить надежность доставки сообщений и оценить достигнутый уровень надежности сервисов.

Introduction

Currently, SOA technology is most popular as the management technology of service and business application level in multiservice networks. The main feature of this technology is that SOA designed for collaboration of heterogeneous loosely coupled applications (or services). In such way a development patterns for SOA focused on principles of accommodation and unification. Now exists only one way to ensure reliability of service is a redundancy. Later, standards for reliable service invocation like WS-Reliability and WS-Reliable Messaging emerged [1], but the problem of achieving reliability of SOA-based Web-services remains unsolved. When trying to address the problem of Web-services reliability, one has to recognize that many of the features that make SOAs attractive, such as run-time service recruitment and composition, conflict with traditional reliability models and solutions.

I. Analysis of reliability characteristics of web services

In the process of solving the problem of ensuring the reliability of Web-services we must remember that the main function, which made the SOA so popular, is a dynamic selection of available service instance.

However this feature is problematic in terms of traditional models to ensuring reliability. In an SOA system, all services are implemented in the so-called "cloud" therefore the calculation of reliability for such systems is a rather difficult task, because it is not known on what elements of the system will be implemented particular service.

Consequently, when the algorithm finding a suitable service instance that satisfies the requirements of the user (including the quality of services (QoS)), agent must take into account the desired level of reliability [1].

As a rule, the reliability of services is pawned on the life cycle relating to the operation and shall not affect the design phase. But experience shows that this often leads to a significant increase the cost of implementing reliable services and reduction of QoS. Separate the following reliability mechanisms [2].

– Detection. The discovery of errors.

– Diagnostics. Search for a device or component in which errors occurred.

– Masking. Disguising of some errors to avoid failures in the system.

– Localization limitations a failure to stop its propagation.

– Recovery. Reconfigure the system with the reason to remove defective units and remove consequences of errors.

Considered mechanisms are usually using the following methods [2].

– Redundancy. The duplication of critical components in a system with the intention of increasing the reliability of the system. This mechanism is often used for the chemical, energy, nuclear and aerospace industries.

– Diversity. Requires having several different implementations of software or hardware specifications, running in parallel to cope with errors that could arise directly from a specific implementation or design.

– Careful reduction. This mechanism is essential in systems where, in the case of system crash is highly unacceptable. Instead, some functionality should remain in the event of a failure. If the operating quality of the system decreases, the decrease should be proportional to the severity of the failures.

– Testing and monitoring. Constant monitoring of the system to check that specifications are crucial in identifying the problem. This mechanism, while very simple, plays a key role in obtaining a fault tolerant system.

– Localization. Errors are contained within some specific execution domain, which prevents errors propagation across system boundaries.

Analysis of the questions of the software reliability have shown that the reliability of services connected not only with the size, combination, complexity of programs but with programming techniques which are used too.

Proposed approach is based on the notion of reliability patterns [3, 4]. A pattern is an encapsulated solution to recurrent software or system problems in a given context, and it can be described using UML or SDL diagrams [3]. Patterns for reliability are support widespread application of best practices and best solutions, offer an effective guideline for software developers that may not have expert knowledge and experience in reliable system development.

The pattern includes of several parts which containing a detailed description of the object, service and links to solutions in terms of reliability. The next step is to determine the achieved level of service reliability through the use of the described template [5].

The pattern should contain the section which described in Fig. 1 and Fig. 2. Particularly Fig. 1 depicts an example of a class diagram for the "Service status" pattern. The main function of this pattern is to detect errors in a system by acknowledging the reception of an input within a specified time interval.

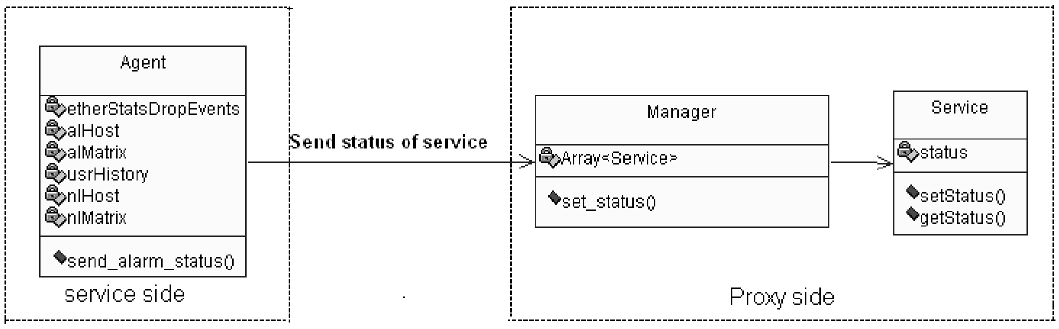

Fig. 1. Class diagram for "Service status" pattern

In the "Send status of service" pattern, the "Sender" (Agent) in conjunction with the "Timer" constitutes the "Monitoring System" (Manager), and the "Receiver" (Service) in conjunction with the Acknowledger entity constitutes the "Manager". "Sender" is responsible for contacting the "Manager". Every time when the "Agent" has sent message to the "Service", the "Timer", that is responsible for counting down the timeout period every time an input is provided to the "Manager", is activated. "Timer" does the countdown waiting for confirmation of successful reception of the message. After each successful confirmation "Agent" activates "Manager".

The sequence diagram corresponding to the algorithm described the interaction of objects, "Agent", "Manager", "Service" are shown in Fig. 2.

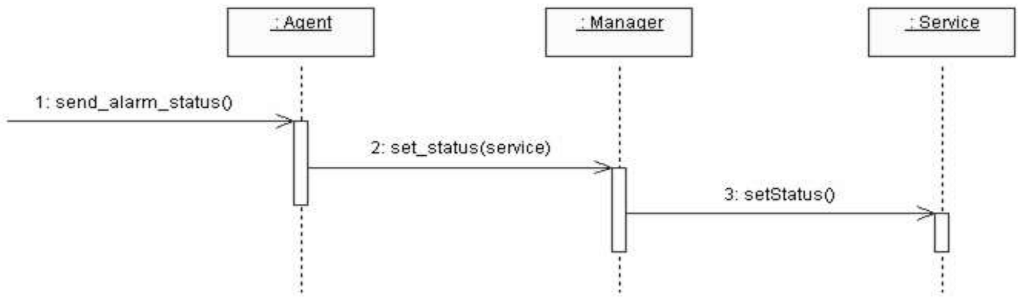

Fig. 2. Sequence diagram for "Service status" pattern

If the timeout period of the "Manager" expires for N consecutive times without receiving an acknowledgment from "Service", "Manager" detects an error on "Service" and notifies "Agent".

A priori validation of reliability patterns provides an estimation of the level of reliability that can be achieved before the actual system is implemented. A priori validation can be performed using the consequences and the failure/fault coverage of a reliability pattern. The consequences of a reliability pattern describe the advantages and disadvantages of using the pattern. This information can be used a priori to compare patterns. A widely used criterion is the amount of computational resources required by the pattern [1, 2].

The failure/fault coverage of a pattern, instead, is described as the number of failures that can be identified, masked, contained, or recovered with its use. This information can also be used for a priori selection of a pattern. For instance the N-Modular Redundancy (NMR) pattern can detect ($N-1$) and mask ($N-2$) faults. Fig. 3 illustrates the structure of NMR.

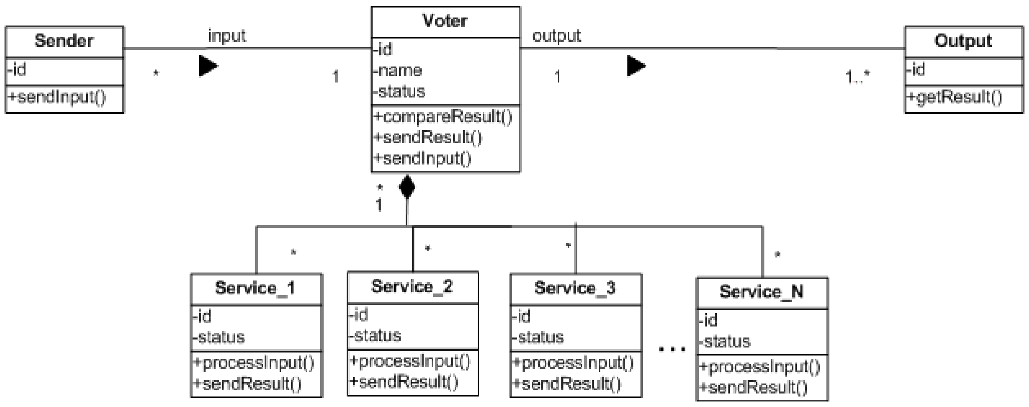

Fig. 3. Class diagram for NMR patterns

The evaluation of pattern consequences and coverage can permit to compare functionally equivalent services a priori (i.e., before they are invoked) on the basis of the level of reliability provided by the corresponding patterns.

II. Reliability evaluation

To assess the reliability of services provided by the patterns proposed to use a cluster testing [4]. Block diagram of proposed solution is depicted in Fig. 4.

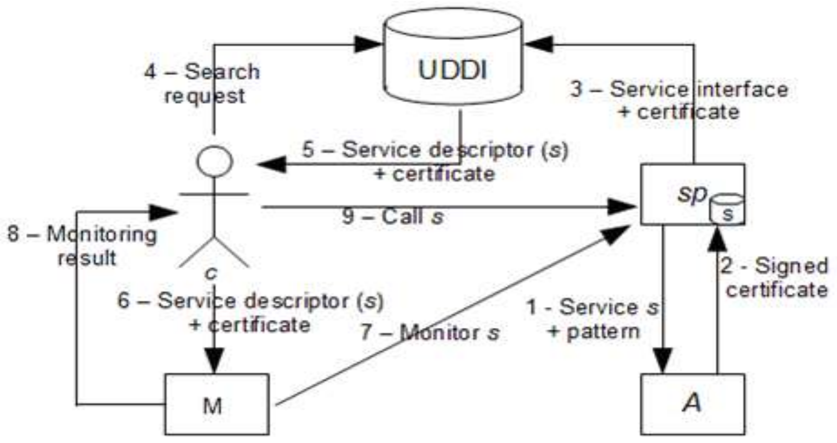

Fig. 4. The block diagram of cluster testing reliability

Our service invocation process enhanced with reliability certification is composed by two main stages. In the first stage (Steps 1-2), "Agent"(A) reliability certificate to a service provider (*sp*) based on a service implementation s and a reliability pattern. In the second stage (Steps 3-9), upon receiving the certificate for the service CS_n, *sp* publishes the information about reliability together with the service interface in a repository. Then the "Client"(C) searches the repository and compares the reliability information of the available services. Once the client has chosen a service, it will ask to "Manager"(M) to confirm its validity. "Manager" checks that the corresponding monitoring rules hold and returns a result to c. If the result is positive "Client" proceeds to call the service.

CS_n consist from n services ($S_1, S_2, ..., S_n$) where S_i can be atomic or composite services. Assume that services $S_{11}, S_{12}, ..., S_{1m}$ are functional equivalents of S_1 from CS_n.

During the period when S_n exploits one set of services can be registries many new services. Therefore, voting service sets the initial scale of 0 to services that are just registered in the system and which have not yet been assigned to the scales. Voting service determines the appearance of errors by comparing the weighted results of testing the service with the results of the majority of alternative services. The discrepancy would indicate an error. Suppose that service reliability S in time point t is equal $R(S,\Delta t)$. In the next period of time Δt, k inspections will be successfully executed and f inspections will have disarrangement in results thus reliability $R(S,t+\Delta t)$ during the time $t+\Delta t$, will be equal to:

$$R(S,t+\Delta t)=\frac{M-k}{M}R(S,t)+\frac{k}{M}R(S,\Delta t), \qquad (1)$$

where M – general number of tests.

Different scenarios can have different execution characteristics, significance of which will determine their weight.

Scenarios are sequences of actions related to each other by four operators: serial, parallel, and the cycle is a logical operator. In scenario can run these types of actions, as the transition from one event to another, performing the specified actions, sending and receiving data, the implementation of subscenario. Consequently, the reliability of data, events, actions, and subscenario will be also affect the reliability of scenario. In general, the whole composite service may consist of multiple execution sequence of scenarios.

Reliability of service is determined by the reliability of all the scenarios that are invoked during the execution of service:

$$real_{system}=\frac{(\Sigma(w_i * real_{scenario_i}))}{\Sigma w_i}, \qquad (2)$$

where w_i – number of launches of $i-th$ scenario.

III. Pattern reliability to different scenarios and composite services

In a service flow, compositional structures describe the order in which a collection of Web-services are executed. There are four types of basic compositional structures: sequence, branch, loop and parallel [6]. The reliability of pattern of these compositional structures is introduced in the following:

– Sequence. The patterns in a sequence structure are executed one by one. The sequence structure is reliable if all of its sub-task reliable. The composed reliability of a sequence structure is calculated by:

$$p=\prod_{i=1}^{n}(1-p_i), \qquad (3)$$

where n is the number of sequential tasks and p_i is the failure probability of the $i-th$ task.

– Branch. In the branch structure, only one branch will be executed for each execution. The composed reliability for pattern is calculated by:

$$p=\sum_{i=1}^{n} b_i(1-p_i), \tag{4}$$

where n is the number of branches, p_i is the failure probability of the $i-th$ branch, and b_i is the execution probability of the $i-th$ branch $\sum_{i=1}^{n} b_i = 1$.

– Loop. The composed invocation reliable of the loop structure is calculated by:

$$p=\sum_{i=1}^{n} l_i(1-p_i)^i, \tag{5}$$

where p_i is the failure probability of the task in the loop structure, l_i is the probability of executing the loop for i times, n is the maximum looping times, and $\sum_{i=1}^{n} l_i = 1$.

– Parallel. All tasks are executed at the same time in the parallel structure, where each branch has an execution probability of 1. The invocation failure probability of the parallel structure is calculated by:

$$p=\prod_{i=1}^{n}(1-p_i), \tag{6}$$

where p_i the probability that the $i-th$ parallel branch will fail.

These basic compositional structures can be nested and combined in an arbitrary way. For calculating the aggregated reliability of a service flow, we decompose the service flow to the basic compositional structures hierarchically. As shown in Fig. 5, firstly, the reliability of the basic compositional structures CS_1 and CS_2 are calculated by using the corresponding formulas which introduced above.

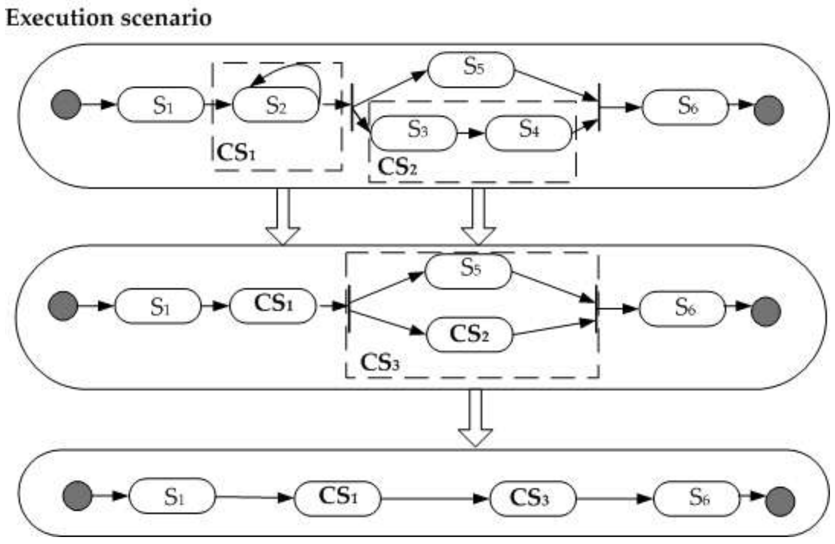

Fig. 5. Reliability of composite services

Then, the reliability of CS_3 is calculated by employing the reliability of S_5 and CS_2. Finally, the reliability of the whole service flow can be obtained by using Equation 3 and the reliability of S_1, CS_1, CS_3, and S_6.

When aggregating the reliability of the service components, we assume that all Web-service component failures are independent, this assumption is reasonable, since Web- services are usually deployed on separate severs of different organizations. The physical and electrical isolation ensures that Web-service failures are independent. However, in some special cases, reliability of Web-services may have correlation (e.g., two Web-services running on the same server, error propagation, etc.).

Conclusion

Today reliability is a key concern in most modern distributed systems. The SOA paradigm, which supports runtime selection and composition of services, makes it difficult to guarantee a priori the reliability of a process instance. In this paper, we presented a technique based on reliability patterns and proposed model for calculation reliability of composite services. At the same time, issues of reliability and QoS were not considered.

In our work, we designed patterns to assess a posteriori reliability of services. We are currently extending our approach to support a wider range of reliability models. By the above approach, designers of the service-oriented systems are able to predict reliabilities of the systems early at the architecture design phase. Moreover, after the system released, the prediction of system reliability can be dynamically updated when the performance of the service components is changed. In addition, we are working to integrate other types of evidence reliability.

Reference:

1. *Zhu K., Duan Z., Wang J.* Quality of service in web services discovery // IEEE Symposium on Advanced Management of Information for Globalized Enterprises, 2008. – P. 1-5.

2. *Scholz A., Buckl C., Kemper A., Knoll A., Heuer J.* Ws-amuse - web service architecture for multimedia services // In Proc. 30th Int'l Conference Software Eng. (ICSE'08), 2008. – P. 703–712.

3. *Grassi V., Patella S.* Reliability prediction for service-oriented computing environments // IEEE Internet Computing. – 2006. – Vol.10, No.3. – P. 43–49.

4. *Kop C.* System reliability and metrics of reliability RMONv2 [Электронный ресурс] // Air Power Australia. – 2011. – Режим доступа: http://www.ausairpower.net/Reliability-PHA.pdf.

5. *Дуравкин Е.В.* Управление сервисами средствами SNMP и RMONv2 [Электронный ресурс] // Проблеми телекомунікацій. – 2011. – № 3 (5). – С. 105 – 110. – Режим доступа: http://pt.journal.kh.ua/2011/3/1/113_duravkin_rmon.pdf.

6. *Wang W.-L., Pan D., Chen M.-H.* Architecture-based software reliability modeling // Journal of Systems and Software. – 2006. – Vol.79, No.1. – P.132–146.