Security Service Granularity

by
Spyridon Antakis

October 19, 2008

Abstract

The paper will present Service-Oriented-Architecture (SOA) as a possible solution to an IT strategy and it will define *service granularity.* Furthermore, it will define *coarse-grained* and *fine-grained* services from a general perspective and from a security perspective. Then, it will present a specific security problem and it will give 2 possible solution scenarios using security service-oriented approach. After the analysis of the scenarios it will try to answer the following questions: *Is it possible to strictly define security service granularity? Is it better to use fine-grained or coarse-grained security services? What are the advantages and disadvantages in using the fine-grained security services?* Finally, it will conclude that *security service granularity* will always depend on the specific goal we want to accomplish each time. Different design approaches will have different impacts, including always factors such as, *reusability*, *cost* and *complexity.*

1 Introduction

1.1 Service Oriented Architecture

Referring to a Service Oriented Architecture (SOA) system, we simple refer to a system that unifies business processes by structuring large applications as an ad hoc collection of smaller modules called services. SOA is a new and popular paradigm of Information Technology (IT) architecture style. It uses services as building blocks to organize and architect the applications in an enterprise. There are several different ways these services can be built, put together, and offered. SOA simple divides the IT support into components in a way related to integration and rebuilding of processes. Each component will have a very well defined purpose and can be reused whenever functionality of that kind is needed. A prerequisite is, however, that the components are not integrated, or coupled, in such a way that a change in component makes another stop functioning; the components need to be loosely-coupled. So, how *loosely-coupled* the components must be and what must be the *granularity level* of a component, in a *security service architecture*? In general, SOA is an IT-architecture style that does not have to follow any strict rules; is therefore not tied to any particular programming languages, vendor, product or standards body. It is only important to realize that SOA services are a strict computer to computer affair; humans do not enter data directly into SOA services, but rather use some kind of interface application which in return calls, one or more SOA services. Each service take some kind of input, does something with it, according to a set of rules and produces some kind of output.[1][2][3]

1.2 Service Oriented Modeling Assets

Before moving on to the section that defines the service granularity level, let first take an overall view of the Service Oriented Modeling Framework (SOMF). This work structure is chiefly a high-level map depicting the various components that contribute to a successful service-oriented modeling approach. It illustrates the major elements that identify the *what to do* aspects of a service development scheme. These schemes could be also considered to a security service system architecture design. All three major formations that service-oriented modeling framework introduces they could be applied also to the security service approach and lead to well-defined service oriented security systems.[4]

Atomic Service: An indivisible software component that is too granular and executes fewer business or technical functionalities. An atomic formation is also a piece of software that typically is not subject to decomposition analysis activities and its business or technological functionality does not justify breakdown to smaller components. (Figure 1)

Composite Service: A composite service structure aggregates smaller and fine-grained services. This hierarchical service formation is characteristically known as coarse-grained entity that encompasses more business or technical processes. A composite service may aggregate atomic or other composite services. (Figure 2)

Service Cluster: This is a collection of distributed and related services that are gathered because of their mutual business or technological commonalities. A service cluster both affiliates services and combines their offerings to solve a business problem. A cluster structure can aggregate atomic as well as composite formations. (Figure 3)

Figure 1: *Atomic Service*

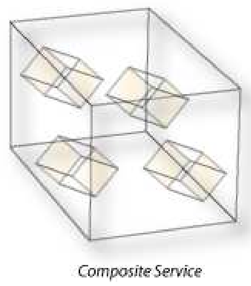


Figure 2: *Composite Service*

2 Service Granularity

2.1 What is Service Granularity

If we strictly define *service granularity* in terms of functionality, then we refer to the breadth of functionality each service encompasses, usually known as the *service abstraction level.*[1] However, service granularity classification could be reflected in two other different interpretations. In the terms of data granularity, that reflects the amount of data that is exchanged with a service and in terms of business value

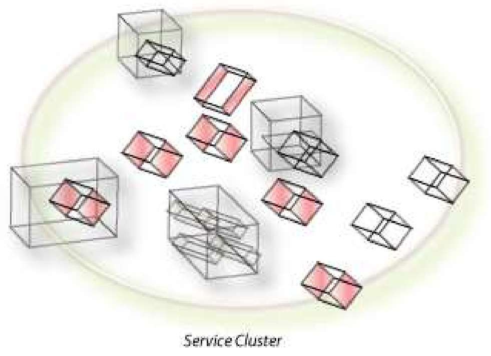


Figure 3: *Service Cluster*

granularity of a service, that indicates to which extent the service provides added business value.[4] In this paper, we will only present service granularity in terms of functionality. *Service granularity* generally refers to the size of a service, but to strictly define granularity is quite complex since it cannot draw on theoretical groundings. Usually, for the description of granularity of a service the terms *coarse-grained* and *fine-grained* are used. *Coarse-grained* term is referring to systems of large components and *fine-grained* to systems of small components. The fact that services should be *coarse-grained* is often postulated as a fundamental design principle of service oriented architecture (SOA).[3] However, there will always be a complexity restraint on how large the services can be and still be manageable. Therefore, its not easy to define an optimal size for a service, a consideration about the complexity needed to be evaluated and re-eavaluted many times, at different points. Hence, service developers are forced to a trade-off in spatial extent.

2.2 What is better, Coarse-grained or Fine-grained Services

It is important to understand what is wrong with building an SOA composed entirely of *fine-grained* services or entirely of *coarse-grained* services. Using LEGO blocks as a metaphor for SOA we will try to illustrate the answer to this question. Implementing SOA solely from *fine-grained* services is like trying to build something only from the smallest LEGO size possible, the 1x1 blocks with a single bump on the top. Sure, each block has the right interface in that you can connect it to other 1x1 blocks, but with this approach it will be extremely difficult to build anything. Similarly, if you think about having only *coarse-grained* services, it would be like having a box of LEGOs shaped like houses and cars. Sure, they have bumps on them, and they do look like what you want, but they are not particularly reusable, and they are entirely useless for building anything other than houses or cars. Thus, you need to have the appropriate assortment of services ranging from *fine-grained* to *coarse-grained* in order to address your particular business problems.[8]

2.3 Service Granularity Matrix

At first view it might seem that all *atomic* services are *fine-grained* and *composite* services *coarse-grained*, this is not necessarily the case. Since granularity is a measure of interaction and business appropriateness, and atomicity is a measure of process decomposition, it is quite possible to have *coarse-grained atomic* services and *fine-grained composite* services. As a way of explaining how this might come about, consider (Figure 4). Evaluating this matrix, one could have *coarse-grained* services that are in fact *atomic* because there no way to decompose them further. Perhaps they originated from a mainframe and the lowest level of granularity you can get is still too *coarse.* This does not make the service *composite.* Likewise, consider a case where a service has been built to provide a small part of a large interaction, and in this consideration it is *fine-grained*, yet the service itself is composed of other services at the same level of *fine granularity.* This might especially be the case in situations where the services are supporting business processes in multiple contexts, such as *fine-grained* telephony services that are themselves composed of *atomic* services exposed from legacy assets. The chart also implies that *fine-grained* services are highly reused, but this is not really always the case, just as *coarse-grained* services are not guaranteed to be business oriented. One can even argue that while *fine-grained* services may be more reusable, if they're too *fine-grained*, that reduces their reusability, because they might tend to have too narrow use, just as services that are too *coarse-grained* are so constrained.[9]

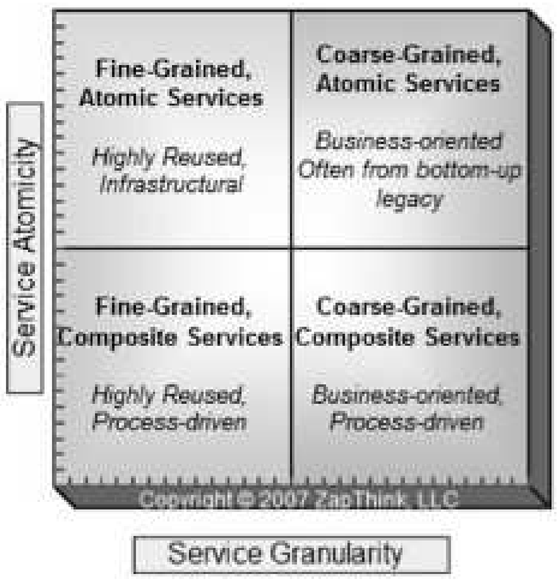

Fine-Grained, Atomic Services
Highly Reused, Infrastructural
Coarse-Grained, Atomic Services
Business-oriented Often from bottom-up legacy
Fine-Grained, Composite Services
Highly Reused, Process-driven
Coarse-Grained, Composite Services
Business-oriented, Process-driven
Service Atomicity
Service Granularity
Copyright © 2007 ZapThink, LLC

Figure 4: *The Service Granularity Matrix*

3 Security Service Granularity

3.1 In General

According to SOA philosophy, applications can not be in charge of security because traditional approaches to security are no longer sufficient enough. Barriers may be good for security, but they get in the way of businness. That does not mean that security can be compromised to meet the business goals. Traditional security approaches assumed and took advantage of barriers, but the SOA is lowering of traditional barriers to reuse- application boundaries, technology boundaries and enterprise boundaries. Thus, security has to be implemented through services with a way that will keep services as open as possible and easy to use. The idea of a security is in some ways similar to the idea of an application service and in some ways different. Like an application service , a security service should be usable by any application; technology differences should not be a barrier. This different approach of implementing security is the source for many questions, that probably have more than one answer. In our case, some interesting questions would be, *are the security concerns the same?, are we able to define an optimal security service granularity level?, what kind of restrictions we will face and what are the advantages and disadvantages for using different security service granularity level?*

3.2 Security Concerns

We can classify security concerns into two groups: a) *Functional aspects of security* and b) *Nonfunctional* *aspects of security.* *Functional* aspects of security are standard in the sense that they exist even with traditional applications as well. In comparison with the *nonfunctional* aspects of security which are nonfunctional in the sense that they do not directly relate to security. Instead, they are required to make sure that a security solution works well in enterprise setting. Below, we briefly present the security concerns of those two groups.[10]

3.2.1 Functional Security Concerns

- *Authentication* - Verifying identity of users.
- *Authorization* - Deciding whether or not to permit action on a resource.
- *Data confidentiality* - Protecting secrecy of sensitive data.
- *Data integrity* - Detecting data tampering and making sure neither the sender nor the receiver can deny the message they sent or received.
- *Protection against attacks* - Making sure attackers do not gain control over applications.
- *Privacy* - Making sure the application does not violate the privacy of the users.

3.2.2 Nonfunctional Security Concerns

- *Interoperability* - This concern is specific to SOA, where different security solutions must not break compatibility of services that are otherwise compatible.
- *Manageability* - This concern is bigger for SOA, as a security solution needs to protect many different services.
- *Data confidentiality* - This concern is common for any security solution. Either for SOA or for traditional application development, the complexity reduces adoption of any security solution.

3.3 Coarse-grained Security Services vs Fine-grained Security Services

Previously, (in section 2), we presented the LEGO metaphor example trying to explain and make it easier for the reader to understand, that is not always possible to decide if it is better to use *coarse-grained* services instead of *fine-grained* services or the other way around. So, as we are trying to look services from the security perspective, the same philosophy applies to *coarse-grained* security services and *fine-grained* security services. In order to define the security service granularity level of a system and whether it is better to use *coarse-grained* security services or *fine-grained* security services, we will have first to consider and evaluate factors, such as: *the scope of the system,what is the goal, who has the goal and how high or low-level is that goal.*[5] After all, *"No one view is best for all"*, and no security volume fits every organization and it is up to organization's view on security IT-Infrastructure to set the bounds.[6] In order to achieve this, some major inhibitors will have to be considered in the designing process, such as, *incompatibility of different computers, operating systems and application standards, different data definitions and different technical standards in several applications and in various parts of the organization.*[6]

4 Presentation of a specific security problem

4.1 Why a specific security problem

In order to understand better the impacts of using the security service approach, this paper analyzes a specific security problem and gives possible solution through different scenarios. *Why a specific problem*? The reasons for using a specific problem, were already explained in a way (in section 3.3). It is impossible to define security service design the same way for every problem, each problem will be approached and solved with different criteria, depending always on the goals of the security system. Thus, the conclusions, the impacts and the considerations for implementing the security service system, will be different for each problem. The goal of this paper is to make the reader understand and evaluate some possible impacts of a security service approach, by analyzing a specific security problem.

4.2 Description of the problem

Imagine, an enterprise company that develops software applications.This company has three different departments developing software applications that customers are actual using. How the security service, that is responsible for the authorization is going to be implemented, *coarse-grained* or *fine-grained*? What will be the actual impact of using different level of security granularity. What are the *advantages* and *disadvantages* in using each different scenario? Afterwards two scenarios will be presented as a possible solution. Scenario A, as a *coarse-grained* approach and Scenario B as a *finer-grained* approach. Both of them will use a kind of role-based access control. Those two scenarios will be over simplified and that because they are intending only to make the reader think and evaluate some of the constraints. It is sure that real life IT security problems are more complex and more difficult to analyze.

4.3 Scenario A

4.3.1 Analysis

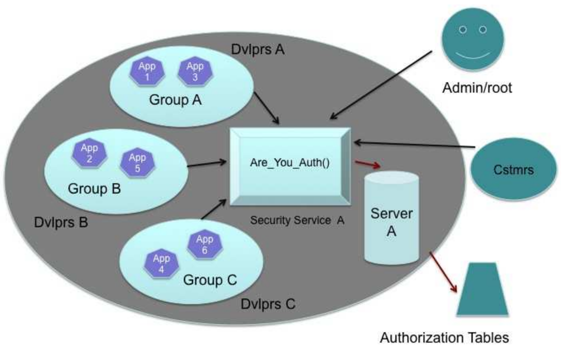
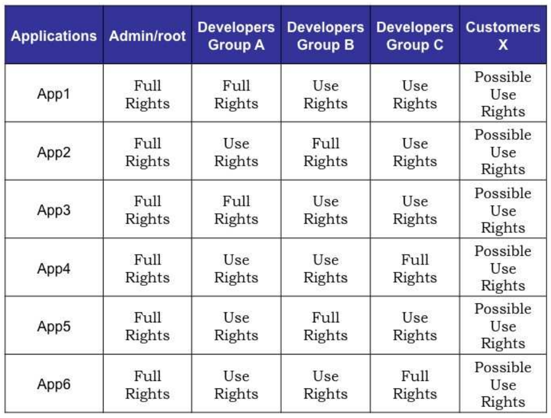


Figure 5: *Scenario A*

In scenario A, we implement authorization through a security service called **Are_You_Auth**(). In terms of functionality, this is a very coarse grained service because, it can check usernames and passwords, it checks access rights and it grants access rights. The prototype of this service is the following,

- **Are_You_Auth**(user, pass, role, app).

So, if *developers*, *customers* or *admin/root* want to access an application then they must use the security service **Are_You_Auth**() and give the proper input: *name, password, role* (customer,admin, developer) and the *application* they want to have access. After that, the security service will handle the request and it will give access to the requester, but only with the pre-configured role based assigned rights on the specific application. If we make a hypothesis, that *admin/root* must have the *full rights*[1] to every application, *developers* must have the *full rights* only on the applications they develop and a *use right* to the others and that *customers* must have only *possible use rights*[2] , then we will have the following table with role-based authorization. (Figures: 5 & 6)



Applications	Admin/root	Developers Group A	Developers Group B	Developers Group C	Customers X
App1	Full Rights	Full Rights	Use Rights	Use Rights	Possible Use Rights
App2	Full Rights	Use Rights	Full Rights	Use Rights	Possible Use Rights
App3	Full Rights	Full Rights	Use Rights	Use Rights	Possible Use Rights
App4	Full Rights	Use Rights	Use Rights	Full Rights	Possible Use Rights
App5	Full Rights	Use Rights	Full Rights	Use Rights	Possible Use Rights
App6	Full Rights	Use Rights	Use Rights	Full Rights	Possible Use Rights

Figure 6: *Scenario A - Authorization Table*

4.3.2 Problems

If we look carefully in scenario A, we will realize that there are some serious problems that this solution have to deal with. There would be at least two levels of granularity that we must take carefully under consideration, the granularity of the functions of the security service and the granularity of the role distribution. Choosing to assign the users rights through roles, will make also some other problems appear. For example, what if one of the *developers A* must have access and *full rights* on *application 4*, for development reasons? That *application* is owned and handled by *developers C* group, but *developer A* has not this role, thus is not possible to have the *full rights*. And how about a *customer X* that has the *use rights* on an *application*, but needs also to interact with that application, this will not be possible, because the *customers* role has only *use rights* assigned. In general, we understand that using this coarse-grained security service to check the authorization only based on roles, we will always depend on the roles. If we want to have access on an application with only a specific right, it is impossible. The only solution is to create a different role or change the rights assigned to an entire existent one, but this

[1]Full rights - having administration rights on the application

[2]Possible use rights - may have may not have the use rights on the application

is not efficient at all. And it becomes more complicated, because until now we assumed that *user* parameter should only refer to physical persons and the *"app"* parameter should only refer to system applications. What about other security services that must use other security services? After all, that is the case in SOA. For example, imagine another security service **Crypto_Password**() that decrypts or encrypts the password that the **Are_You_Auth()** security service takes us an input, in order to make the comparison with authorization tables data. Then, the things could become really complex.

4.4 Scenario B

4.4.1 Analysis

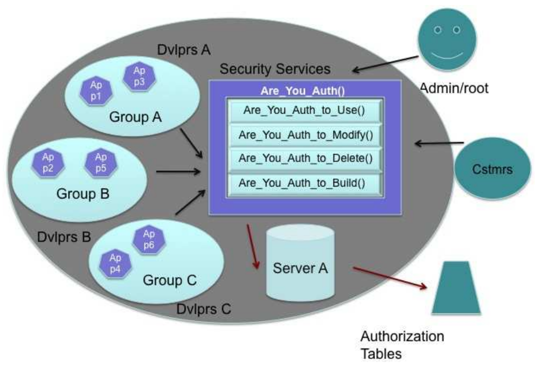


Figure 7: *Scenario B*

In scenario B, we implement authorization through four finer-grained security services that are aggregated to the previous coarse-grained security service which scenario A used. So, the security services used now are,

- **Are_You_Auth_to_Use**()
- **Are_You_Auth_to_Modify**()
- **Are_You_Auth_to_Delete**()
- **Are_You_Auth_to_Built**() and
- **Are_You_Auth**()

The prototypes of the 4 new services are the following,

- **Are_You_Auth_to_Use**(user, pass, app)
- **Are_You_Auth_to_Modify**(user, pass, app)
- **Are_You_Auth_to_Delete**(user, pass, app)
- **Are_You_Auth_to_Build**(user, pass, app)

for the **Are_You_Auth**() security service the prototype is the same

- **Are_You_Auth**(user, pass, role, app)

Now, if *developers*, *customers* or *admin/root* want to access an application they could use the security service **Are_You_Auth**() and give the proper input: *name, password, role* (customer,admin,developer) and the *application* they want to have access. Then a check based on their *role* will be performed and they will have access with role based rights. But now, also they could use one of the finer-grained security services to have only a specific permission right on the application, giving as input only: *name, password* and the *application* they want to have the access. In this way they will have the right to use, modify, delete and build an application without having the *full rights*. Thus, as an example a *developer A* could make use of the security service **Are_You_Auth_to_Build**() so that he has authorization in building an application for *developers C* without however to have the *full rights* on this application or to change his role. Another example could be, a *customer* that makes use of security service **Are_You_Auth_to_Modify**() in order to have authorization for interaction with an application but again without been assigned to a new role.

4.4.2 Problems

In scenario B, the numbers of security services increased, and a finer-grained approach was presented. Nevertheless, also in this approach there are several problems left, that need to be solved. First of all, we may have solved the limitations of the authorization checks that were made only in a role based way, but still a consideration must be made about how the rights are going to be assigned beyond the role based approach. Also, maybe some restrictions must be applied in the use of the new aggregated security services, because the use of the one it could be meaningless without the right to use the other. For example, if a *developer* has the authorization to use security service **Are_You_Auth_to_Build**() will probably need also the right to use security service **Are_You_Auth_to_Modify**() on that application.

5 Conclusions

It is not 100% clear how we can define *security service granularity* in a general way. Different views have different impacts on *performance*, *reusability* and *cost* for Enterprise Companies. Measure will always apply in relation to the *security services* and to the number of interactions required to accomplish a specific goal. From a general point of view, we can say that *fine-grained* security services are better than *coarse-grained*. We should have in mind that building a security system from *fine-grained* security services is more *complex*, *expensive* and *difificult*, but there will always be a *high-level of reusability* in return. From the other side, building a security system from *coarse-grained* security services maybe is *easier*, *cheaper* and *less complex*, but there will always be an important *lack of reusability.* If they were no constraints about the time, the money and the actual effort of designing, modeling, and implementing a security system, then some of the disadvantages that appear with the *fine-grained* approach, would not really matter so much. Unfortunately, in the real world that is impossible. Time, money and effort will always be factors that will have to be taken under a serious consideration.

References

[1] *Pierre Reldin, Peter Sunding,* (2007), "Explaining SOA Service Granularity", Linkping University.

[2] *Szyperski C.,* (2003), "Component Technology - What, Where, and How?", Proceedings of the 25th International Conference on Software Engineering.

[3] *Raf Haesen, Monique Snoeck, Wilfried Lemahieu and Stephan Poelmans,* (2008),"On the Definition of Service Granularity and Its Architectural Impact"

[4] *Michael Bell,* (2008),"Service-Oriented Modeling: Service Analysis, Design, and Architecture"

[5] *Cockburn A.,* (2001),"Writing Effective Use Cases", Boston: Addison-Wesley cop.

[6] *Weill P., Broadbent M.* (1998),"Leveraging The New Infrastructure", Boston: Harvard Business School.

[7] *Anderson B., Hagen C., Reifel J., Stettler E.* (2006), "Complexity:customization's evil twin. Strategy and Leadership", 19-27.

[8] *ZapThink::Service-Oriented Architecture Expertise, Advisory, and Influence* "Right-Sizing Services" by Ronald Schmelzer, `http://www.zapthink.com/report.html?id=ZAPFLASH-20051115` (Accessed on: 02/10/2008)

[9] *ZapThink::Service-Oriented Architecture Expertise, Advisory, and Influence* "The Service Granularity Matrix" by Ronald Schmelzer, `http://www.zapthink.com/report.html?id=ZAPFLASH-200783` (Accessed on: 02/10/2008)

[10] *Kanneganti R., Chodavarapu P.* (2008), "SOA Security", 'Manning'