ABSTRACT
Although the Service Oriented Architecture (SOA) concept has been accepted by many companies and organizations, it is still facing the existing challenges in security requirements and problems. One of the disputed security aspects in this architecture is Access Control. In this article, not only, are we trying to show the problems of traditional access control models when encountered by basic principles of SOA, but also, we are going to present a new language based access control (LBAC) that uses the concepts of languages and set theory as its formal bases. In this model an access language will be defined for each resource of organization consisting of functions and services, and a language will be assigned to each subject. Then, using its fundamental authorization rule, the model only allows the access to subjects that have a string of the language of requested resource. Using a sample scenario, we will show that proposed model can achieve the main goal of access controls which is making correct decisions about issuing permissions. Furthermore, this model shows better conformity with primitives such as interoperability, composeability and encapsulation from the security point of view. A small scaled implementation has also been presented.

KEY WORDS
Access Control, Case Study, Service Oriented Architecture, Composeability, Encapsulation.

1. Introduction
By separating the concerns, for the purpose of reusability, SOA has been able to decrease the cost of Information Technology in organizations [1]. One of the instances of reusability is service composable [1]. Based on composable concept, a composite service will implement its business logic using some other services. SOA has emphasis on composable of existing services and reusing them to perform new businesses [1]. On the other hand, access control as a security mechanism attempts to prevent unauthorized access to resources [2]. In order to balance these extremes, an appropriate access control model is required in such a way that responds to security requirements and has conformity with SOA primitives.

Three access control models exist [2]: Discretionary (DAC), Mandatory (MAC) and Role Based (RBAC). Since, these models have been designed for centralized systems, without taking into account the requirements of SOA; in this research we will show encountering the basic SOA principles such as composeability, reusability and interoperability, they will cause some security vulnerabilities.

The rest of the paper is organized as follows. The next section presents a sample scenario showing vulnerabilities in existing access control models encountering basic SOA principles. In section 3 we will explain our proposed model. In section 4 the sample scenario will be examined by our model and its advantages over other models will be illustrated. An implementation of our model using web services and SOAP standard will be explained in section 5. The last section concludes this paper and suggests some future works.

2. Sample Scenario
Suppose a composite service like Money Transfer that is composed from Deposit and Withdraw services. Let withdraw service be more secret than Deposit service, also user U has been allowed only to use this composite service in the system. In Figure 1, by sending its request to composite service, the requester issues a request in order to transfer money from one account to another.
This transformation, consequently causes the invocation of inner services which is according to encapsulation principle [1], the requester should not be aware of this composite logic.

Now, we will examine this scenario for three existing models.

2.1 Discretionary Access Control

In discretionary access control model, there is an authorization list in the access matrix per each user with several triples <s,o,a> -one triple per each authorization- which shows that the subject s is allowed to perform action a on object o [2].

Based on autonomy principle [1] regarding Deposit and Withdraw services, using DAC implies that there should exist access entries for Deposit and Withdraw services in the access matrix, as well as the entry for Transfer service. The philosophy behind adding those extra entries to the access matrix is to enable composed services authorizing the user's requests being received via composite service by its Identification Number (ID). However, as it has been depicted in Figure 2, this will result in security vulnerability: direct invocation of Withdraw service by the requester.

By abusing this vulnerability, the attacker may become able to charge her account arbitrarily. Thus, reusability through service composition is accompanied by direct and unauthorized use of composed services security threat.

Furthermore, this risk exists that the number of access entries in the authorization matrix exposes the composite logic of Transfer service and as a consequence, the encapsulation principle is being violated.

One situation that can expose the existence of access permission to withdraw service in the access matrix is the support of authority delegation [2]. According to this concept, each subject must be aware of its authorization list to be able to delegate any subset of it to another subject.

2.2 Role Based Access Control

In Role Based Access Control, there exist some roles with a set of permissions assigned to them to access resources [2]. Roles could be assigned to users as well. A user is allowed to access a resource only if she has its permission available in rights of her active roles [2][3]. We could consider two strategies for Transfer scenario. First, we enforce authorization only in composite service level and as a result, Deposit and Withdraw services will trust on Transfer service authorization. Therefore, in rights list of user's role, there will only be the permission to composite service.

So because of the lack of access permissions to composed services in user's authorizations, the risk of direct access to these services is not possible. However, new security vulnerability has emerged. In the case of intrusion to the composite service, in addition to exploiting composed services directly and unlimitedly, other unauthorized services may be called through these composed services illegally. As it is shown in Figure 3, if the user would be able to access Deposit service in her local domain, it may be possible for her to call the services of other banks, say the same service in inter-domain scenario. This vulnerability has occurred as a natural result of the lack of access control among services and trusting the prior service authorization. It is worthy to be mentioned that only subjects are being authorized against objects in RBAC [3].

The second strategy is direct authorization by composed services, themselves. In order to make this real, it is necessary to append the access permission of Deposit service to the rights list of user's role. Therefore, the composed services check user's permissions themselves for the purpose of responding to the composite service request. This idea does not have the first strategy vulnerability, however, adding access permission to Deposit service in the rights list of user's role, not only does it have the direct access vulnerability, but also, simultaneously, all the subjects with the same role will be able to call this service which in turn, will aggravate the situation. On the other hand, the existence of access permission to Deposit and Withdraw services in the rights list of user's role, is accompanied with the risk of composite logic exposure and as a result the encapsulation principle may be violated.
2.3 Mandatory Access Control

The mandatory model, classifies subjects and objects based on secrecy and integrity [2]. Also, a partially ordered SET (POSET) relation is being defined on these classes [2]. Subjects and objects are being classified in TS, S, C and UC levels, and the users' authorization is based on the aforementioned POSET relation [2]. The dominance relation is defined as follows: TS>S>C>UC [2].

In accordance with our scenario, since Deposit service requires higher level secrecy, it is necessary for Transfer service to possess higher level security class, in order to be able to call Deposit service. Thus, if Deposit service is in TS security class, Transfer service will be in TS, too (Figure 4). On the other hand, according to dominance relation [2], in order for the requester to be able to access Transfer service, its security level must be greater than the requested service namely TS. By keeping this description in mind, it is necessary that the requester be in highest security level which is in contradiction with minimal authority [4] and minimal duty [3] principles.

3. The Proposed Model

The proposed model in this research is an authorization based access control model [5] that resembles two-level security architecture approach [6] in such a way that two models exploited the multi-level architecture idea. The authorization in this model is done using the languages and sets theories [7] concepts.

3.1 Definitions and Notations

We shall use in our notations the shorthands \( \leftrightarrow \) and \( \rightarrow \) as in Boolean logic and also \( \forall, \exists, \neg, \in, \notin, \cup, \subseteq, \emptyset \) as in set theory.

**Definition 1 Subject**: any user or service that could issue a request to a resource.

**Definition 2 Resource**: any software unit that is responsible to execute a request which is granulated as follows:

- A specific function of a service
- A specific service
- A set of services same level services

Per each resource, a language will be defined that specifies its access to that resource.

**Definition 3 Access Language** (\( L_i \)): a unique infinite regular language [7] that is defined for resources and a subset of it will be assigned to favourite subjects. The regularity property of access language is for the decidability of our fundamental access rule, and the infinity property is for the possibility of unlimited generation of access strings in authorization process.

**Definition 4 Merge Operator** (\( \Delta \)): an n-ary operator that is defined on languages and strings as follows:

\[
(L_\Delta = \Delta_{i=1}^k (L_1, L_2, ..., L_k)) \quad (1)
\]

\((\forall w \in L_i; w \in L_{j}, i \in \{1,2,3, ..., k\})\).

The merge operator is equivalent to Union operator in sets and languages theories [7] and by executing it on a set of languages, a new language will be created such that can generate strings of all its composer languages. Therefore, if \( w_i \in L_i \) and \( w_j \in L_j \) then \( w_i, w_j \in L_\Delta \). This operator has four overloading as follows:

\[
w_{ij} = \Delta(w_i, w_j) : \text{Union } w_i \text{ and } w_j. \quad (2-1)
\]

\[
w_{ij} = \Delta(w_{ij}, L_{ij}) : \text{Generates a string from } L_{ij} \text{ and union it by } w_{ij}. \quad (2-2)
\]

\[
L_{ij} = \Delta(L_i, L_j) : \text{Union } L_i \text{ and } L_j. \quad (2-3)
\]

\[
L_{ij} = \Delta(L_i, w_j) : \text{Generate } L_{ij} \text{ for } w_j \text{ and merge it by } L_i \quad (2-4)
\]

Where \( L_i, L_j \) be access the languages and \( w_i, w_j \) be two strings.

**Definition 5 Decompose Operator**: a n-ary operator that is defined on languages and strings as follows:
\[ L_q = \nabla^k(L_{1}, L_{2}, \ldots, L_{k}) \rightarrow \forall w \in L_q : w \not\in L_{i}, i \in \{2, 3, \ldots, k\}. \] (3)

In the above expression, the decompose operator separates \( L_{2} \) to \( L_{k} \) from \( L_{1} \) in such a way that there will be no string of the newly generated language that belongs to \( L_{2} \) to \( L_{k} \).

Therefore, if \( w_{i} \in L_{1}, w_{j} \in L_{j} \) then \( w_{i} \not\in \nabla(L_{i}, L_{j}) \). Also, this operator is equivalent to \textit{Difference} operator in sets and languages theories [7].

So, the expression 4 will be true about merge and decompose operators:

\[ L_q = \nabla(L_{1}, L_{2}) = L_{1} - L_{2} \]
\[ L_{\Delta} = \Delta(L_{1}, L_{2}) = L_{1} \cup L_{2} \] (4)

The decompose operator has four overloading as similar to merge operator:

\[ w_{ij} = \nabla(w_{i}, w_{j}) : \text{Removing } w_{j} \text{ from } w_{i}. \] (5-1)

\[ w_{ij} = \nabla(w_{i}, L_{j}) : \text{Removing string of } L_{j} \text{ from } w_{i} \] (5-2)

\[ L_{ij} = \nabla(L_{i}, L_{j}) : \text{Removing } L_{j} \text{ from } L_{ij}. \] (5-3)

\[ L_{ij} = \nabla(L_{i}, w_{j}) : \text{Removing } L_{j} \text{ from } L_{i}, \text{ where } L_{j} \text{ is the language of } w_{j}. \] (5-4)

We implied by \textit{Removing}, the separation of \( w_{j} \) string from composite string \( w_{i} \) in the 5-1 expression. For example, if \( w_{i} = aa \cup bb \), \( w_{j} = bb \) and \( \cup \) operator be \textit{union}, removing \( w_{j} \) from \( w_{i} \) yields \( aa \).

**Definition 6 Function Language** \( (L_{f,s_f^k}) \): is an access language for \( k \)-th function of \( j \)-th service in \( i \)-th level such that by possessing this language or a string belonging to it like \( w \in L_{f,s_f^k} \), the access to this resource is made possible.

**Definition 7 Service Language** \( (L_{s_i}) \): is an access language for \( j \)-th service in \( i \)-th level such that by possessing this language or a string belonging to it like \( w \in L_{s_i} \), the access to this resource is made possible.

According to SOA layers from IBM point of view [8], services are either atomic or composite. For each atomic service, the access language is computed by expression 6 using merge operator on the access languages of its functions:

\[ L_{s_i} = \Delta^k_{k=1}(L_{s_i}) \] (6)

\( k \) is the number of current service functions.

Where, \( L_{s_i} \) and \( L_{f,s_f^k} \) are service and function access languages respectively.

In case of composite services, the access language would be selected from unique languages as similar to functions. So, the access language of composite service and its composed services are disjoint, also possessing a string of this language, permission to composed service will not be granted at all.

By defining authorization tuple for composite service, the permission to its inner functions is only granted to the composite service itself that will exclude any other access attempts from outside (refer to definition 9 for authorization tuple).

**Definition 8 Level Language** \( (L_{l}) \): is an access language for \( i \)-th level such that by possessing this language or a string belonging to it like \( w \in L_{l} \), the access to all services in this level is made possible. Access language for each level could be computed from expression 7:

\[ L_{l} = \Delta^m_{k=1}(L_{l_k}). \] (7)

\( m \) is the number of services in current level.

**Definition 9 Subject Authorizations:** accesses or authorizations of subjects are determined by a tuple consisted of subject identity, an access language and an access string as in expression 8:

\[ [\text{Subject}_i, L_{\text{subject}_i}, w_{\text{subject}_i}]: L_{\text{subject}_i} = \Delta(L_{\text{level}}, L_{\text{service}}, L_{\text{function}}). \] (8)

Where, \( L_{\text{level}}, L_{\text{service}}, L_{\text{function}} \) are any subset of level, service and function languages respectively; each of these languages may be nullable[7].

The \( w_{\text{subject}_i} \) string in access tuple consists of \textit{delegated} accesses that may be created by the subjects’ members of the system, using merge and decompose operator over languages or strings. The \( L_{\text{subject}_i} \) is the access language of the subject which is defined using merge and decompose operator over access languages sets solely by security administrator. It is worthy to be mentioned, we suppose the \textit{integrity} in all circumstances, is guaranteed.

**Definition 10 System Security Administrator:** a user possessing the access language of the whole system or a string of it:

\[ [U_u, L_u, w_u]: L_u = \Delta^m_{i=1}(L_i), \] (9)

for all \( l_i \) in organization, \( w_u \in L_u \).

**Definition 11 Unauthorized Subject:** a subject who has no access tuple in the system.
**Definition 12 Guest Subject:** a subject with an access tuple that its access language has just epsilon string and its access string is epsilon too. Thus, its access tuple is as:

\[ [U_{\text{g}}, \varepsilon, \varepsilon] : \varepsilon \text{ is an empty string.} \]  

\[ (10) \]

We distinguish unauthorized subject from the guest one in our model. Although the guest subject has an epsilon access language and string, since the access language of the resource might be also nullable [7], the existence of \( \varepsilon \) enables the usage of resources having the nullable access language. Thus, as opposed to unauthorized subject, the guest subject has a level of controlled permissions in the system.

This distinction between unauthorized and guest subject is beneficial for some scenarios in SOA where we want to grant controlled and limited access to a one-time user without requiring the permanent registration of her complete information profile [9]. Moreover, the guest user may be a user whose all permissions have been already revoked.

### 3.2 Preliminaries

In this section we are going to present our model preliminaries.

**Preliminary 1:** The merge and decompose operator have to be dual of each other. By duality we mean, selecting appropriate languages (refer to Preliminary 4), the expression 11 is true for all the overloads of these two operators:

\[ \forall (L_i, L_j) \subseteq L \text{, } \Delta(L_i, L_j) = \Delta(L_j, L_i) = L_i \]  

\[ (11) \]

If we grant \( L_i \) to a user with \( L_i \) access language, and afterwards revoking \( L_i \), the remaining access languages is precisely equal to its previous value namely \( L_i \) (right and left side of expression 11). Nevertheless, by re-granting a revoked access to the subject's access language, there would be no extra unforeseen access for that subject (middle and right side of expression 11). This preliminary reassures us that using merge and decompose operators does not cause any inconsistent languages in the system. Because of the utmost importance of this preliminary, we are going to prove an application of it in theorem 1.

**Theorem 1:** By unlimited revocation and granting of access via merge and decompose operators; no unforeseen access language will emerge.

**Proof:** According to expression 4, we are going to rewrite expression 11 as follows:

\[ ((L_i \cup L_j) - L_i) = ((L_i - L_j) \cup L_i) = L_i \]  

\[ (12) \]

Using distribution property of union and difference [7] operators we will have:

\[ (L_i \cup L_j) - L_i = (L_i \cup L_j) \cap L_i = L_i \cup (L_j \cap L_i) = L_i \cup \emptyset = L_i \]

\[ (12-1) \]

\[ (L_i - L_j) \cup L_j = (L_i \cap L_j) \cup L_j = L_i \cap (L_i \cup L_j) = L_i \cap \emptyset = L_i \]

\[ (12-2) \]

The expression 12-1 show the equality of left and right side of expression 12 and the expression 12-2 shows the equality of the middle and right side of 12. Because of the equivalence of expression 11 and 12, the theorem is proven.

**Preliminary 2:** To be accessible to all subjects in the system, the resource must have a nullable access language:

\[ \varepsilon \in L \text{ (} R_i \text{ is accessible for all } S_i \text{) } (13) \]

Where, \( L_i \) is an access language to resource \( R_i \). This preliminary is for the purpose of supporting guest subjects concept mentioned in definitions 12.

**Preliminary 3:** Any change to resource access language is only possible via merge and decompose operators and this is only done by system security administrator. As an example, it may be possible to add a new function to a service (extendability). So, the access language of the service has to be changed. Furthermore, adding a new service to a specific level of the services (scalability) requires the modification of that level's access language.

**Preliminary 4:** All functions' and composite services' access languages have to be bilaterally disjoint:

\[ \forall i,j : L_{R_i} \cap L_{R_j} = \emptyset \]  

\[ (14) \]

Where, \( L_{R_i} \) and \( L_{R_j} \) are access languages. By bringing this preliminary we are ensuring the least privilege and separation of concerns principles [3]. Since, the access languages are bilaterally disjoint, subjects having the access language \( L_{R_j} \), only are permitted to use resource \( R_j \) and access to \( R_i \) is impossible (separation of concerns). Also, to make sure the obedience from minimal authority...
principle, it is sufficient to grant the fewest subsets of access languages to the subject (least privilege).

For example, we have selected the following language set for our model:

\[ L_i = a^{p_i} \text{ if } n > 0, \text{ where } \Sigma = \{a\} \]

and \( p_i \) is i-th prime number starting from 3.

Where, \( n > 0 \) if the languages are nullable.

Any regular and infinite language that satisfies definitions and preliminaries of our model can be used as an access language.

After selecting and assignment of languages to resources and subjects based on the model definitions and preliminaries, it is made possible for us to perform the access control process. The access control logic behind LBAC model is to answer a key and decidable question in language theory: "Verifying the membership of a string to a regular language" [7]. The access control decision engine makes its decisions based on the belonging the access string of the requester or authorization token [6] to the resource access language.

### 3.3 Rules

LBAC model has only one fundamental rule for authorization purposes. Having access tuple \([S_i, L_{R_j}, w_{S_i}]\) for a subject and \( L_{R_j} \) as an access language of \( R_j \), expression 16 shows this access rule:

\[
\left( \exists w \in L_{S_i} \land w \in L_{R_j} \right) \lor \left( w_{S_i} \in L_{R_j} \right) \iff R_j \text{ is accessible for } S_i.
\]

Please note that \( L_{S_i} \cap L_{R_j} \neq \emptyset \) or \( L_{S_i} \subseteq L_{R_j} \) would not necessitate allowing access in expression 16, but by having \( L_{S_i} \supseteq L_{R_j} \), the access will be always permitted.

The aforementioned rule in expression 16 is the basis for access control in our proposed model. Whenever a request has been issued for a subject in order to access a resource, this request will be delivered to the access control decision engine. This engine that is our Policy Decision Point (PDP) [6], checks the membership of the requester access string to the resource language and makes proper decisions. If the string belongs to the access language of the resource, request will be permitted, otherwise denied.

### 4. Verifying the proposed model by sample scenario

Due to two-level access control strategy regarding composite services, Transfer service as a subject has the access to Deposit and Withdraw services in its access tuple.

The access tuple of the Transfer service has been indicated in expression 17:

Access Tuple of Transfer Service=
\([S_{\text{Transfer}}, \Delta (L_{\text{Withdraw}}, L_{\text{Deposit}}), \epsilon]\)

(17)

It is sufficient for a subject to be granted the access to composite service, as has been shown in expression 18:

Access Tuple of Requester User =
\([U, L_{\text{Transfer service}}, w]\]

(18)

Therefore, the access language of the composite service is not equal to access language in its tuple:

\[
L_{\text{Transfer Service}} \neq \Delta (L_{\text{Withdraw}}, L_{\text{Deposit}}).
\]

The composite service according to its access tuple invokes its composed services (regardless of user's access tuple). In other words, in this scenario, composed services only permit access via their composite services and therefore, direct access to them will be impossible. It is because, the subjects have not access language or string of composed services at all (Figure 5). Formally, we say:

\[
L_{\text{Transfer service}} \cap L_{\text{Withdraw (local Bank)}} = \emptyset \]

(19-1)

\[
L_{\text{Transfer service}} \cap L_{\text{Deposit (local Bank)}} = \emptyset \]

(19-2)

Figure 5. Composeability principle immunity from security threats
Since, $L_{\text{Transfer service}}$ is the access language of the requester's tuple, expressions 19-1 and 19-2 shows that, having the language of composite service, it is impossible to use inner services directly. According to expression 14 (refer to preliminary 4), the expression 20-1 and 20-2 will exists:

\[
L_{\text{Requester}} \cap L_{\text{Withdraw in Bank}} = \emptyset \quad (20-1)
\]

\[
L_{\text{Requester}} \cap L_{\text{loan}} = \emptyset \quad (20-2)
\]

Therefore, using our proposed model not only does the composability principle remain immune from mentioned threats, but also it seems that it does not have vulnerabilities to new threats.

5. Implementation

It is possible to implement the security logic of SOA in three ways: as a part of service logic, using security services, as a part of Enterprise Service Buss (ESB) [10]. In a case study, we have implemented the access control logic of our model using security service whose conceptual modeling has been depicted in Fig 6.

Figure 6. Implemented security services for proposed model

The position of this service in the interaction model of SOA has been shown in Fig 7.

Figure 7. The position of authorization model in SOA interaction model

Also, we suggest using relational database in order to maintain access tuples as is shown in Figure 8 and we have successfully implemented it for our sample scenario.

Figure 8. Access database

The proposed model has been implemented using .Net framework [11] and SOAP standard [10] over web services. Each requester puts her authorization token in a SOAP header message before calling her favourite service. The issued SOAP message that is consisted of parameters of the destination service's function in the body, and the authorization token in the header, will be received by PDP. By extracting authorization token from SOAP message header and retrieving the access language of desired resource from access repository (database), a Turing Machine [7] in access control decision engine of PDP, determines if the access string of requester belongs to the access language of the resource that as a consequence the request will be permitted.

Since the main implementation logic is based on SOAP standard protocol, we claim that our implementation is platform independent. Requester-side implementation (using C#.Net) of our sample scenario is as follows:

TranfserService transServiceInstance = new TransferService();
transServiceInstance.Header = GetLBACSOAPHeader(subjectRequester, requestedObject);
transServiceInstance.Transfer (DepositAccountId, WithdrawAccountId, amount);

In PDP by extracting the header of SOAP message, decision making about permission is being done:

Authorization (SOAPHeader);

In the body of Authorization function, we have implemented a Turing Machine to verify the membership of access strings to the resource languages:
Bool IsAuthorized =
IsStringBelongToLanguage(
header.AuthorizationToken, resourcelanguage);

6. Conclusion and Future works

The strength of Service Oriented Architecture is in using principles like: Reusability, Interoperability, Composability and encapsulation. Current access control models, are accompanied by several vulnerabilities in some scenarios. Making use of formal models which are more compliant with SOA principles have shown to be more reliable. The proposed access control model shows a better compatibility with the requirements of SOA. Table 1 which is created based on the Sample Scenario in our research clarifies this compatibility.

Table 1 Comparison between proposed and conventional models based on SOA indices

<table>
<thead>
<tr>
<th>Model</th>
<th>Composability</th>
<th>Encapsulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC</td>
<td>The direct access of composed services security threat</td>
<td>The risk of composite logic exposure</td>
</tr>
<tr>
<td>RBAC</td>
<td>The direct access of composed services security threat</td>
<td>The risk of composite logic exposure</td>
</tr>
<tr>
<td>MAC</td>
<td>The use of same level services with composed services threat</td>
<td>No vulnerabilities found?</td>
</tr>
<tr>
<td>LBAC</td>
<td>Safe</td>
<td>Safe</td>
</tr>
</tbody>
</table>

It is clear that the vulnerabilities of composeability as being shown in Table 1, also affect reusability. Because of the novelty of proposed model, more studies and experimentations is required in many fields. Our present research line is to show that appropriate using merge and decomposes operators, the proposed model could be able to support RBAC and DAC models, and however, MAC needs farther and more precise studies. Proposed model has the following advantages and properties. Some of them are in the research line of the authors and will be proven in future works. Supporting:

1- Interoperability principle
2- Reusability principle
3- RBAC, DAC and MAC models
4- ad-hoc users
5- Authority Delegation

6- Inter organization communication as a result of begin authorization based
7- Avoidance of Identity based Access Control problems (e.g. FldM)[5]

Model performance evaluation in variant environments requires relevant experiments and analyses. Finally, implementation by SOA security standards like SALM is in process currently by the authors of this research.

References