Modeling Access Control Characteristics of Components in Uniframe Framework Using Logic Programming

1Dhownya Bhatt, 2Dr. Ekata Gupta

1 Research Scholar(CSE)
Mewar University, Chittorgarh

2 Associate Professor, Department of Mathematics
Krishna Institute of Engineering and Technology

Abstract
Uniframe is an approach that facilitates the interaction of heterogeneous software components. In a Uniframe framework, the entire working of the integrated system is transparent though the individual components that have the access control guards can be identified. Only certain resources in the contained resources are protected that are selected by the system Integrator. This is beneficial because the components as a whole need not to be protected instead which will help minimizing the work of the system integrator. Components are modeled with access control behavior and thus they become protected resources. The most important task is to establish functionality between the protected and the unprotected components when the whole system is integrated and put to perform. Using logic Programming to model the access control characteristics of the components is one of the most efficient way of guarding the system. Prolog returns the truth value of the query as resulting from the original database. The research proposed in this paper focuses on modeling the components with access control properties thereby making them making any ordinary contained resource into a protected resource using Prolog. Also this process will involve modeling user priorities, the access control guards, the derived and the dynamic attributes and access control policy of the chosen logic programming language-Prolog. Towards the end of this paper, the component specification for the concrete components mentioned in this paper is explained and the General Terminal is coded through Prolog paving way for the establishment of model driven access control architecture. The dependability of system on the access policy is verified.

Keywords – Modeling, Component, Behavior, Predicate, Privilege, access guards, Prolog

I – PROLOG IN MODELING COMPONENT BEHAVIOR
The Prolog is a logic programming language that uses structures, variables and constants to construct a statement. The structures form the most unique parts of the Prolog as they are the propositions of predicate calculus represented as - functor((parameterlist))A variable is a string of letters or digits or underscores that begins with the uppercase. A constant is an atom or integer. An atom is a string of letters or digits or underscores that begins with the lowercase. The statement declaration can be headed or headless Horn clauses. Headed Horn clauses represent definite relationships while Headless Horn clauses define the facts. While modeling the facts and are stated in the form of a “Proposition” that usually is returned with a YES or NO. When there are two or more queries in the process then unification is attempted by the system to find the true value and the Prolog identifies the variable that results when the query is YES.Prolog uses two types of processes for matching query to facts and rules to relationship. One is forward chaining in which system starts with processing the actual facts to a resolution that derives queries. The other is the backward chaining that starts with a query to resolve a matching proposition that leads to a fact in the database or a rule. Components that are modeled with security properties are put up in a list to separate them from the other unprotected components. Prolog too supports a datastructure called “List” that is a sequence of any number of elements put within square brackets and separated by a comma. List based predicates are used in this research as a base to form the Prolog modeling of all components taken for study.

While the access control behavior of the component is presented as a logical predicate, Prolog uses one predicate for one function in the component’s interface. This helps to model only the access related behavior of the function and not the whole-full component. This saves the time of system integrator as stated above and also increases the efficiency of the system. This also helps to frame an “access policy” that states the limits of any user and determines what component function he can execute.Also the concrete components are described by the rules supported by Unified Meta Model(UMM) the contained and the resource that is required should be added to specification when the component is searched. This can be represented through a table that specifies the UMM auxiliary attribute will possess an access control panel to represent the name of resources as contained only or protected. This helps in component matching based on access properties when the Uniframe Resource Discovery Service (URDS) tries to find contained and protected resources.
For instance consider a concrete component Employee Record server (ERS) as in [Dho01] that holds the information about Admin and Employees in an organization. The admin is modeled with priorities to add new employee and view details about his workers like ongoing project status etc., The employee on his level is able to view the details of his personal bio-data and information about the project in which he is currently associated with. Also the employee has to be restricted from viewing the details of his colleague or peeping in the admin privileges. But it has to be ensured that the employee receives all important information even when he is not in the office or working at a different site or on a holiday. The table below is formatted as per UMM specifications along with auxiliary security attributes.

<table>
<thead>
<tr>
<th>Concrete Component: EmployeeRecordServer</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Component Name: EmployeeRecordServer1</td>
</tr>
<tr>
<td>(ii) Component Subcase: EmployeeRecordServerCase1</td>
</tr>
<tr>
<td>(iii) Domain Name: Corporate</td>
</tr>
<tr>
<td>(iv) System Name: Employee Information</td>
</tr>
<tr>
<td>(v) Informal Description: Provide a employee information system in an organization to the Admin.</td>
</tr>
</tbody>
</table>

(vi) Computational Attributes:

<table>
<thead>
<tr>
<th>Inherent Attributes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: N/A</td>
</tr>
<tr>
<td>Version: version 1.0</td>
</tr>
<tr>
<td>Author: N/A</td>
</tr>
<tr>
<td>Date: N/A</td>
</tr>
<tr>
<td>Validity: N/A</td>
</tr>
<tr>
<td>Atomicity: Yes</td>
</tr>
<tr>
<td>Registration: N/A</td>
</tr>
<tr>
<td>Model: N/A</td>
</tr>
</tbody>
</table>

Functional Attributes:

<table>
<thead>
<tr>
<th>Function description: stores personal and professional employee information and coordinate requests to other additional servers to process the query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm: N/A</td>
</tr>
<tr>
<td>Complexity: N/A</td>
</tr>
<tr>
<td>Syntactic Contract</td>
</tr>
<tr>
<td>Provided Interface: IEmployeeManagement, IEmployeeRecord</td>
</tr>
<tr>
<td>Required Interface: IEmployeeManagement, IEmployeeRecord, IServiceRecord, IPerformanceEmployment, Interactions</td>
</tr>
</tbody>
</table>

(vii) Cooperation Attributes

<table>
<thead>
<tr>
<th>Preprocessing Collaborators: GeneralTerminalCase1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Postprocessing Collaborators: EmployeeRecordServerCase1, EmployeeInformationServerCase1, EmployeePromotionServerCase1</td>
</tr>
</tbody>
</table>

(viii) Auxiliary Attributes:

<table>
<thead>
<tr>
<th>Mobility: No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
</tr>
<tr>
<td>Access Control</td>
</tr>
<tr>
<td>Contained Resources: *.gsi</td>
</tr>
<tr>
<td>Protected Resources: *.gsi</td>
</tr>
<tr>
<td>Fault tolerance: L0</td>
</tr>
</tbody>
</table>

(ix) Quality of Service

<table>
<thead>
<tr>
<th>QoS Metrics: throughput, end-to-end delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS Level: N/A</td>
</tr>
<tr>
<td>Cost: N/A</td>
</tr>
<tr>
<td>Quality Level: N/A</td>
</tr>
</tbody>
</table>

Table 1 – Concrete Component ERS represented through Unified Meta Model

II- COMPONENT INTERFACE MODELING WITH PROLOG

In component modeling, each method is expressed as a predicate. Initially one method is called in a component and when this call gets completed, one or more methods on another component are called as a chained process. But the predicate of the originating method is evaluated depending on the result call of other methods. For illustration, consider the predicate
model writeEmployeeRecord on concrete component Employee Record Holder Server EIS in [Dho01]. For this particular predicate to functionally evaluate, all calls made individually to General Terminal, Employee Information Server, and Access Privilege Server must be YES or in other words the call must succeed. The calls to all of these servers will get a positive response when the complete chain as mentioned is executed successfully. This also includes the fact that the information about an employee can be READ or read partially. But the WRITE operation is completely not permitted and any command to perform this should be returned with NO and the process should be terminated.

saveEmployeeRecord(P,E,ER):
- (1)
  saveGeneralRecord(P,E,ER),
  saveEmployeeInformation(P,E,ER),
  saveAccessPrivilege (P,E,ER)

The saveAccessPrivilege shows that access authorization to be cleared, so the above predicate is modified with an access guard for protecting write invasion to employee information record. The access privilege server is one of the concrete component.

saveEmployeeRecord(P,E,ER):
- (2)
  guard(P, [re,S],[category,gsi],write),
  saveGeneralRecord(P,E,ER),
  saveEmployeeInformation(P,E,ER),
  saveAccessPrivilege (P,E,ER)

One advantage of using PROLOG over other logical programming language is that the predicates for both concrete and protected components can be combined into one database of PROLOG that can represent the whole system as such.

III – USER PRIVILEGES MODELING

This is one of the most important component modeling as the user’s authorization and authentication is decided by the access control model through the security attributes. It is important to note that access control model should also be able to identify the privileges possessed by each of the system users. Propositions of predicate calculus represented through functor (parameter list) is used here creating a user ID, Project group, role played in a particular project such as database lead, software analyst, Testing Engineer. If dbhatt as in [Dho01] is a software developer then the predicate is represented as,

authenticate(dbhatt,credential(accessID(dbhatt),role([softwaredeveloper]),group([employee])))

(3)

authenticate(nala, credential(accessID(nala), role([peon]), group([staff])))

(8)

The security modeling proposed through this research paper in this research will utilize the above predicate and the information furnished in two states. When the model and the system has to identify the principal in the system, the predicate will be as,

principal(AID,P) - authenticate(AID,P)

(4)

This refers to accessing information of a principal (the main user) through a means like the accessID through a query in PROLOG – principal(dbhatt, P) The second state is to identify the user itself whether or not the name is in the database of authenticated users.

Employee(E):- authenticate(E,P),
arg(2,P,ROLEPRED),
arg(1,ROLEPRED,ROLE),
member(employee,ROLE)

(5)

Where ROLEPRED indicates the role predicate as mentioned in (3)

Along with the component specification of a concrete component, the authenticity predicates for the principal for a user credential for example dbhatt, an employee who is also a software developer and nala can be given as,

authenticate(dbhatt, credential( accessID(dbhatt), role([softwaredeveloper]), group([employee])))

authenticate(nala, credential(accessID(nala), role([peon]), group([staff])))

principal(AID,P):- authenticate(AID,P)
IV – MODELING OF ACCESS GUARDS OF SYSTEM COMPONENTS

The modeling of access control guards solely depends on the system developer and mostly concerns retrieving dynamic attributes from the corresponding server. Control guards will refer to the access control policy generated through Matching Algorithmic Procedure (MAP) as proposed in [Dho01] to determine authenticity. To complete this particular request, the access control guards should be provided with principal, resource requested and the nature of operation. P, R and O generally is represented in a predicate form,

\[
\text{guard(P,R,O)-dynamicAttributes(P,NEWP,R)}
\]

Here NEWP is the variable that represents the modified set of attributes returned from the dynamic attribute server and this new set of attributes are used now onwards while processing the access request.

Now as the dynamic attributes play an important role in modeling, it is essential to model a portion of data stored in the components. So taking the case of an organizational structure as proposed in [Dho01], the employee should be given access to his own personal details and as well as the details of the working project he is currently associated with. But there has to be certain authenticity done to determine the rightful user. Hence proper modeling of the dynamic security attributes is of much significance. So to enable this, resource naming is the source from where this dynamic attribute is obtained from. This means that the employee information record is linked by the unique employee ID (eID). Also in PROLOG, if the principal ACSID is same as employee ID (eID) then the particular employee’s information will be appended to security attributes through the PROLOG predicate,

\[
\text{EmployeeRecordServer_1} \_ \text{saveEmployeeRecord}(P,E,ER)
\]

V. MODEL COMPOSITION

Model composition of all components to system level components involves unique predicate naming when all the predicates are combined. Both Component name and method name is taken to give the name of the predicate for the individual component method. Hence the chances of same definition of two component predicates will be avoided. Going by this definition, if a new employee record has to be saved, then saveEmployeeRecord if has to be created by employee_record_server_1 will be indicated by the predicate,

employee_record_server_1 \_ saveEmployeeRecord(P,E,ER)

i) Access Policy Modeling in Uniframe

Policies are important as they determine how users can perform their operations. The authenticity and authorization of a particular user completely depends on modeling access control policies in a system. In PROLOG, this action is performed through a single predicate accessControl(P,R,O) where P, R and O represent the principal’s security attributes, resource name, and operation as mentioned above.

accessControl(P,R,O)-arg(3,P,GROUPPRED),arg(1,GROUPPRED,GROUP),

(member(emplyee,GROUP)->
checkemployeeROLES,R,O):checkstaff(ROLES,R,O)
VI. COMPONENT SPECIFICATION FOR CONCRETE COMPONENTS

The concrete components are those which form the base of model driven architecture upon whom the access control properties have to be implemented. Component based Software Development has made these objects that are self-explained in certain environments and accessed to have a public interface in a uniframe with private implementation. Uniframe is one such approach that facilitates the interaction of heterogeneous software components. The concrete components used for illustrating the facts of an access control model in uniframe framework are,

- General Terminal Server
- Employee Record Server
- Employee Information Server
- Access Privilege Server
- Employee Performance Server

For now, only the general declarations of the component specification of each of these are discussed in this research. At a later point of time the actual names and designs designated to each of these components will replace the contents in the symbols [[ ]].
i) General Terminal Server (GT)

- Purpose – this is the login page. The user information is submitted here and is routed through the employee record server. It is common for all and the access privilege is also taken into considerations while given the names and designations to the employee depending on the position held in an organization. This mostly contains the user name which is a unique ID and the information is retrieved from the Employee Record Server (ERS) to this page after the query is processed to YES.

The component specification of a GT is given as,

\[
\begin{align*}
&[[\text{GENERAL\_TERMINAL}]]\text{readEmployeeRecord}(AID,E) \\
&\quad\text{principal}(AID,P), \text{employee}(E), [[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{readEmployeeRecord}(P,E) \\
&[[\text{USER\_TERMINAL}]]\text{saveEmployeeRecord}(AID,E,ER) \\
&\quad\text{principal}(AID,P), \text{employee}(E), [[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{saveEmployeeRecord}(P,E,ER)
\end{align*}
\]

ii) Employee Record Server (ERS)

- Purpose – this is used in employee information storage at various levels. Personal and professional details are managed as a database.

The component specification of an ERS is given as,

\[
\begin{align*}
&[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{readEmployeeRecord}(P,E) \\
&\quad[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{Guard}(P,[[\text{re},E],[\text{category,project}]],\text{read}); \\
&\quad[[\text{EMPLOYEE\_INFORMATION\_SERVER}]]\text{readEmployeeInformationRecord}(P,E); \\
&\quad[[\text{EMPLOYEE\_PERFORMANCE\_SERVER}]]\text{readEmployeePerformanceRecord}(P,E); \\
&\quad([[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{Guard}(P,[[\text{re},E],[\text{category,gsi}]],\text{read}))
\end{align*}
\]

\[
\begin{align*}
&[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{saveEmployeeRecord}(P,E,ER) \\
&\quad((\text{member}(\text{gsi},E)), \\
&\quad[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{Guard}(P,[[\text{re},E],[\text{category,gsi}],[\text{all,all}]],\text{save}); \\
&\quad\text{not member}(\text{gsi},E)), \\
&\quad((\text{member}(\text{ei},E)), \\
&\quad[[\text{EMPLOYEE\_INFORMATION\_SERVER}]]\text{saveEmployeeInformationRecord}(P,E,ER)); \\
&\quad\text{not member}(\text{ei},E)), \\
&\quad((\text{member}(\text{ep},E)), \\
&\quad[[\text{EMPLOYEE\_PERFORMANCE\_SERVER}]]\text{saveEmployeePerformance}(P,E,ER)); \\
&\quad\text{not member}(\text{ep},E)), \\
&\quad((\text{member}(\text{er},E)), \\
&\quad[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{Guard}(P,R,O) \\
&\quad[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{DynamicAttributes}(P,\text{NEWP},R), \\
&\quad\text{accessControl}(\text{NEWP},R,O)
\end{align*}
\]

\[
\begin{align*}
&[[\text{EMPLOYEE\_RECORD\_SERVER}]]\text{DynamicAttributes}(P,\text{NEWP},R) \\
&\quad\text{arg}(1,P,\text{ROLEPRED}),\text{arg}(1,\text{ROLEPRED},\text{ROLE}), \\
&\quad\text{arg}(2,P,\text{GROUPPRED}), \\
&\quad\text{append}(\text{ROLE},[\text{employeeinfo}],\text{ROLE1});\text{append}(\text{ROLE},[\text{ROLE1}]), \\
&\quad\text{append}(\text{ROLE1},[\text{ownerinfo}],\text{ROLE2});\text{append}(\text{ROLE2},[\text{ROLE2}]), \\
&\quad\text{NEWP}=\text{credentials}(\text{AIDPRED},\text{role}(\text{ROLE2}),\text{GROUPPRED})
\end{align*}
\]

iii) Employee Information Server (EIS)

- Purpose – This Server holds transcripts of individual user and his senior employee will be able to access information through this portal.

The component specification of an EIS is given as,

\[
\begin{align*}
&[[\text{EMPLOYEE\_INFORMATION\_SERVER}]]\text{readEmployeeInformation}(P,E) \\
&\quad[[\text{EMPLOYEE\_INFORMATION\_SERVER}]]\text{Guard}(P,[[\text{re},E],[\text{category,ei}]],\text{read})
\end{align*}
\]

\[
\begin{align*}
&[[\text{EMPLOYEE\_INFORMATION\_SERVER}]]\text{saveEmployeePerformance}(P,E,EP)
\end{align*}
\]
iv) Access Privilege Server (APS)

- Purpose - The information regarding the current project the employee works upon and the related details according to his level (Developer/ Project-Head/ Project Manager/ COE) will be displayed. Since access is related to all the other concrete components, it is directly explained in the access control policy framing instead of individually composing it.

v) Employee Performance Server (EPS)

- Purpose – This Server holds transcripts of individual user and his senior employee will be able to access information through this portal.

The component specification of aEIS is given as

\[
\text{[[EMPLOYEE\_PERFORMANCE\_SERVER]}\text{Guard(P,[[re,E],[category,e]]},\text{save)}
\]

\[
\text{[[EMPLOYEE\_PERFORMANCE\_SERVER]}\text{Guard(P,R,O)}
\]

\[
\text{[[EMPLOYEE\_PERFORMANCE\_SERVER]}\text{DynamicAttributes(P,NEWP,R),}
\text{accessControl(NEWP,R,O)}
\]

\[
\text{[[EMPLOYEE\_PERFORMANCE\_SERVER]}\text{DynamicAttributes(P,NEWP,R)}
\text{arg(1,P,AIDPRED),arg(1,AIDPRED,AID),}
\text{arg(2,P,ROLEPRED),arg(1,ROLEPRED,ROLE),}
\text{arg(3,P,GROUppPRED),}
\text{member([sr,AID],R)->append(ROLE,[[employeeinfo],ROLE2];append(ROLE,[],ROLE2)),}
\text{member([ownerinfo,AID],R)->append(ROLE2,[ownerinfo],ROLE3);append(ROLE2,[],ROLE3)),}
\text{NEWP=credentials(AIDPRED,role(ROLE3),GROUPPRED)}
\]

\[
\text{vII. PROLOG CODE FOR GENERAL TERMINAL}
\]

The concrete component is assigned with an additional method so that the chained process goes on uninterrupted as the system is completed with the access control architecture. This unique method of PROLOGis termed as getComponentACModeland to implement this two parameters are required. The component name and the names all the other chain of the other agents has to be provided to the system through which the component will construct and return a string value that contains the Prolog for its particular access behaviour. This is implemented for the General Terminal through Prolog as an example.

```java
public String getComponentACModel(String componentName,
HashtablepostProcessingCollaborators)
{
```
StringBuffer

```java
PrologBuffer = new StringBuffer();

PrologBuffer.append("%....."+componentID+".....\n");
PrologBuffer.append(componentName+
      
      
      
      "_readEmployeeRecord(AID,E)-principal(AID,P)," +
      
      "employee(E),");

String employeerecordServerName = (String)

postProcessingCollaborators.get("employeerecordserver");

PrologBuffer.append(employeerecordServerName);
PrologBuffer.append("_readEmployeeRecord(P,E)\n");
PrologBuffer.append(componentName+
      
      "_saveEmployeeRecord(AID,E,RE)-" +
      
      "principal(AID,P),employee(E),");
PrologBuffer.append(employeerecordServerName);
PrologBuffer.append("_saveEmployeeRecord(P,E,RE)\n");
```

} 

**Table 2 – Prolog representation of General Terminal access behaviour**

It is equally important to verify the functionalities of the system are dependent or not upon the access policy. It is done through querying the database of Prolog coded in table that determines the access control behaviour. The predicates used for this in the General Terminal are P and E representing access ID of the user and the employee ID whose record is retrieved.

employee_record_server_1_readEmployeeRecord(P,E)

employee_record_server_1_saveEmployeeRecord(P,E,RE)

Here RE is the part where information is written into. REis set to the list [gsi,ei] checks if the user is authenticated to perform this particular operation. To elaborate this consider the user privilege predicate,

gerenal_terminal_1_readEmployeeRecord(nala,dbhatt)

which checks if nala, a peon can access the record of dbhatt with a prior declaration of,

    authenticate(nala, credential(accessID(nala), role([peon]), group([staff])))

For this query Prolog returns YES or NO after checking if the user is allowed to access the data or not. Also query can be put to find if or not a particular user can save information to a particular record or not using

gerenal_terminal_1_saveEmployeeRecord(nala,dbhatt,[gsi])

Elaborating this if a particular FO can save more than one information about an employee then the portion that is being written into is represented as,

    General_terminal_1_saveEmployeeRecord(nala,dbhatt,[gsi,se])

These queries test the ability of the system in access controlling the resources and the usage of system resources thereby the system functionality. Queries in programming helps to put to regulated access to the resources before the actual system is deployed to use. Access policies can be changed prior to the implementation of another new policy. So this newly implemented policy will control the system functioning now. This is a way to test the behaviour of the system implemented with access control.

**IX. FACTS in the PROPOSED RESEARCH and FUTURE WORK**

The motto of this research paper is to use the logic programming language Prolog to develop a means to test the system based on its access control properties. Because this is done in prior to the system composition this paves way for static testing of the system and helps to elaborate the functionalities of the system with reference to the access control
properties. While the system functionalities are dealt at the static level, those dynamic properties that change with the nature of access control can be elaborated as a part of the future work to be done. Also Temporal Logic of Actions(TLA) can be used to create a system-based model that will explain the dynamic properties and how they change depending on access control properties can be shall be done as a part of future work.

REFERENCES