

Enhancing Mediation Security by Aspect-Oriented Approach *

Li Yang, Raimund K. Ege, Huiqun Yu

School of Computer Science
Florida International University
Miami, FL 33199, USA
{lyang03|ege|yhq}@cs.fiu.edu

Abstract

Research on mediation techniques to integrate data from heterogeneous data sources has made comprehensive progress. However, mediation poses extensive security problems. Protecting proprietary data from unauthorized access is recognized as one of the most significant barriers to the mediation systems. Neither traditional access control methods are adequate to model the flexible access control requirements, nor are they amenable to manage the evolvable security features of mediation systems. This paper addresses to enhancing mediation security by aspect-oriented approach. The basic functionality components and security concerns are separated, independently specified, and then systematically integrated into a unified model. Our approach benefits in both decreasing design complexity of mediation systems and increasing flexibility and dependability of security enforcement.

Keywords: *mediation system, security, aspect-oriented, specification*

1. Introduction

To gain information from many heterogeneous data sources is the trend for future information system. The mediation [22] task is an extended amalgamation of searching, querying and updating in traditional information systems. Such task can be accomplished via a mediation strategy, i.e. semantic mapping [7, 5] and answering query by source descriptions [21, 11]. In such a mediation strategy, mediators are typically employed to provide an integrated view of information from heterogeneous sources [1, 6]. A mediator provides a mapping of complex models to enable interoper-

ability between clients and sources. One important issue is how to enforce protection for data sources such that every access to a system is controlled, and only those authorized access can take place.

Traditional access control method such as mandatory access control (MAC) and discretionary access control (DAC) [13] are inadequate to reflect the dynamic mediator environment and the flexible access control requirements. Role-based access control (RBAC) [17, 10] models are receiving increasing attention as a generalized approach to access control. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management. However, security concerns are usually scattered across the entire system, which is difficult to design and manage when the target system is large and complex. A promising new approach to constructing systems with evolvable security features is suggested by the work of Aspect-oriented programming (AOP) [12], which addresses separation of concerns in software development by using specialized mechanisms to encapsulate concerns whose behavior crosscuts essential application functionality.

Inspired by the idea of AOP, we propose a method to enhance mediation security in specification level. Based on our previous work [8, 23], we show how to specify the security concerns and apply them to the mediator modular specification in a uniform way. Datalog is used to specify the mediator functional modules, and first-order predicates are employed to specify the security aspects independently. In logic view, the predicate of security aspects can serve as the condition part of the Datalog specification in mediator, which makes it possible to weave the security aspects into the mediator specification.

The rest of the paper is organized as follows: Section 2 explains our adaptive three-layered mediation framework that features an open adornment-based data model; Section 3 constructs the secure mediation framework via aspect orientation; Section 4 is the conclusion.

*Supported in part by the NSF under grants HRD-0317692 and CCR-0226763, and by NASA under grant NAG 2-1440

2. Our Mediation Framework

2.1. A Three-Layered Mediation Architecture

Our mediator architecture organizes sets of intermediate mediators into layers to handle requests from a user which can be any special device or mobile computing unit (as in Figure 1). The mediator sets will play intermediate roles between users and data sources, to help establish streams to and from the heterogeneous data sources. A user can either query or update heterogeneous data sources.

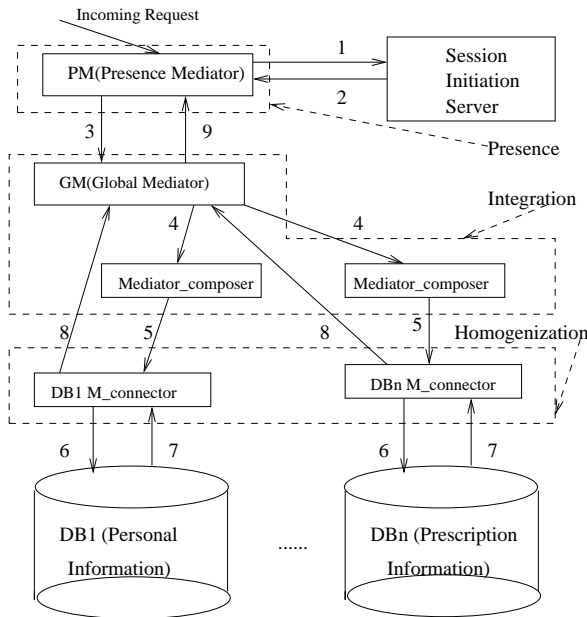


Figure 1. A three-layered mediation architecture

The framework features three layers: presence, integration and homogenization/connector. The upper level is the presence layer that makes the data source seem ever-present to the user and communicates directly with the user. The presence layer is responsible for translating heterogeneous requests from the user into an XML format, extracting the data type of request represented by an XML schema, and translating the response from the XML format into the original user request format. The presence layer makes it feasible that the mediation architecture handles requests from any kind of devices or in any kind of formats via a the two-way format translation. Therefore the work of the underlying layers is encapsulated.

The middle integration layer resolves the schema differences between the user needs and the source availability by schema mapping [3]. The entities in the integration layer are the “Mediator_composers” who are able to decompose the schema if necessary and locate the destination data source for a specific schema via a distributed hash table

algorithm (DHT) [19]. Upon every request from a client, the *session initiation server* coordinates with the “Mediator_composers” group to elect the global mediator. This process of global mediator election dynamically determines the hierarchical structure in the integration layer for each request. This procedure makes the architecture more adaptive to both the network capability and mediator load, and then more efficient for the multimedia data operation, i.e. streaming, than a fixed architecture.

The bottom level homogenization layer contains “Mediator_connectors” that resides on top of actual data sources, and maps the data source schemas to XML schemas. The “Mediator_connectors” stream the actual data or update the underlying data sources upon the request from mediators in the integration layer. They make heterogeneous data sources appear to have a unifying XML schema. As such, we establish an adaptive mediation architecture to handle the two-way data (could be multimedia data) operation (query or update) between the heterogeneous requests and heterogeneous databases.

2.2. Data Model

The primary motivation for mediation technology is to provide support for a broad spectrum of heterogeneous data which are available in different formats. A sound solution to the data integration task requires a clean abstraction of the different formats: any data must be mapped to an *exchange* model from which it is therefore accessible without the use of specific software. Some systems, e.g. SIMS [4] or DISCO [20], or MMM [9], have a fixed application schema like conventional databases. But many systems, e.g. HERMES [2] or Garlic [16], allow for flexible adaptation of the schema as further sources are integrated.

We introduce a *lightweight* exchange model based on XML, enhanced via security (and potentially other) adornments. It is called *the adorned XML model* (AXM). AXM is flexible in data organization, both in the structures that can be described and in the differences in terminology. The security adornment of AXM is essential for the system security. An AXM object has five attributes:

1. *Object ID*. It may be constructed by the mediators to be an expression describing where the object came from. It may also be a pointer to an object in the workspace used to answer the query.
2. *Label* tells what the object represents. Labels are expected to have human-understandable definitions that may be retrieved easily by the user.
3. *Adornment*. Adornment entry identifies the security properties that affect the data processing and system execution. Security adornment indicates a mapping

of principal identities and/or attributes thereof with allowable actions. It is a kind of security policy expression that is often essential in the access control in order to protect resources against unauthorized access. Security adornment plays an important role in the process by which use of resource is regulated according to a security policy and is permitted by only authorized system entities according to that policy.

4. *Type* of its value, either complex type or a simple type like *string*.
5. *Value*, either an atomic value or a set of objects.

With these primitives, it is possible to simulate all the structures that are found in more conventional object-oriented type systems. The adornments can be used not only to define the permissions of the objects in data sources but also to define the roles of the access user.

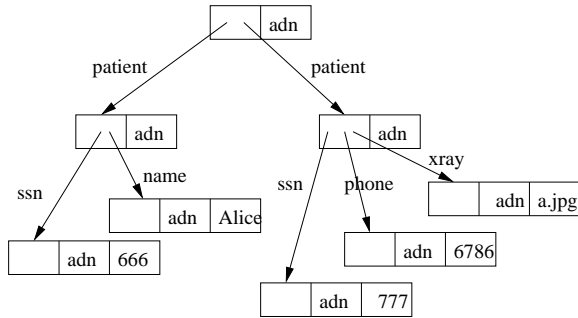


Figure 2. A collection of AXM objects

Figure 2 shows a collection of AXM objects. At the top is a *root* object whose label is *patientGroup*. Its value is a set of *patient* objects, so its type is *complex*. To model semi-structured information sources, we do not insist that data is as strongly structured as in standard database models. For instance, *name* information is sometimes given and sometimes missing.

3. Aspect-Oriented Approach to Enhancing Mediation Security

3.1. The Aspect-Oriented Method

The major issues involved in aspect-orientation for mediation systems are (1) how to separate various concerns, and (2) how to weave aspect models into an integrated system. Issue 1 (*Separation of concerns*) is to separate the functionality modules of mediation systems from the security aspects, which consists of the following steps:

- Identifying basic functionality components in the mediation systems and specifying these components.

- Specifying security requirements.
- Defining the crosscutting section (join points) of the functionality components and security aspects.

Issue 2 (*Aspect weaving*) generates an integrated system by weaving aspect models with the functionality components. The steps include:

- Locating the joinpoints where the functionality components and security aspects interact.
- Defining the behavior of the system in order to enforce security policies on the basic functionality components.
- Integrating aspect models with the functionality components.

3.2. The Component Specification

In our mediation architecture the mediators in the integration layer form the components. Given a set of data sources exported from the homogenization layer, we build mediators to integrate and refine the information. The approach is in the spirit of the declarative specification of mediators in Tsimmis [15]. The query interpretation process is analogous to expanding a query against a conventional relational database view. We will use an example to illustrate the mediator specification and the query interpretation against the mediator specification.

Let us consider two mediators called *med* and *max* that export objects with label *patient*. The *patient* objects fuse information about patients that have the same social security number and are exported by the sources s_1 , s_2 and s_3 . In particular, if source s_1 contains a patient and his name, the exported *patient* object contains the corresponding *name*. If s_2 contains the x-ray examination for the patient and s_3 contains the address information, then the *xray* and *addr* sub-objects are also included in the *patient*. A specification consists of *rules* that define the view exported by the mediator. Each rule consists of a head followed by a : – and a tail. The head describes view objects, whereas the tail describes conditions that must be satisfied by the source objects. In general, the heads and tails are based on patterns of the form {<object-id adornment label value>}.

The specification of the *patient* object appears in two mediators specification (“MS1” and “MS2”); each rule in the specifications describes the contribution of the sources:

```
(MS1) (R1.1)
<pid(S) (adn R) patient {<(adn R) name N>}>@med :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) name N>}>@s1
  AND role_assign() AND check_permission()
  (R1.2)
<pid(S) (adn R) patient {<(adn R) xray X>}>@med :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) xray X>}>@s2
  AND role_assign() AND check_permission()
```

```

(MS2) (R2.1)
<pid(S) (adn R) patient {<(adn R) addr A>}>@max :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) addr A>}>@s3
  AND role_assign() AND check_permission()
(R2.2)
<pid(S) (adn R) patient {<(adn R) xray X>}>@max :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) xray X>}>@s2
  AND role_assign() AND check_permission()

```

The first rule declares that:

- **if** there is a pair of *binding* s and n for variables S and N (variables are identifiers starting with a capital letter) such that s_1 contains a *patient* top-level object that has a *ssn* sub-object with value s and a *name* subobject with value n , the user was assigned a role after authentication by *role_assign()* predicate, and the data access permission was check against source s_1 by *check_permission()* predicate,
- **then** mediator *med* exports a *patient* object, with object-id *pid(s)*, that has a *name* subobject with value n and a unique system-generated object-id.

The semantics of the second rule in *MS1* (“R1.2”) and in the two rules in *MS2* are defined accordingly. As stated in Section 2.1, each mediator has the potential to be elected as the global mediator and take care of the authentication job.

To illustrate how to interpret the query against the mediator specification, assume that a client wants to retrieve all data of *patient’s* with object-id *pid(‘666’)*. The query can be expressed as:

```

(Q1) <pid(‘666’) (adn R) patient PT> :-
  <pid(‘666’) (adn P) patient PT>

```

The object pattern (or patterns in the general case) that appears in the query tail is evaluated against the object structure of the mediator in exactly the same way that the mediator specification rule tails are evaluated against the object structures of the *mediator_connector*. The object pattern of the query head does not include the usual “@” notation because it is implied that the objects described by the query head refer to the result that will be materialized by the client.

After evaluation the tail of sample query (Q1) against the head of the rules in (*MS1*) and (*MS2*), (Q2), (Q3) and (Q4) are sent to the sources s_1 , s_2 and s_3 respectively.

```

(Q2)
<pid(‘666’) (adn R) patient {<(adn R) name N>}> :-
  <(adn P) patient {<(adn P) ssn ‘666’> <(adn P) name N>}>@s1

```

```

(Q3)
<pid(‘666’) (adn R) patient {<(adn R) xray X>}> :-
  <(adn P) patient {<(adn P) ssn ‘666’> <(adn P) xray X>}>@s2

```

```

(Q4)
<pid(‘666’) (adn R) patient {<(adn R) addr A>}> :-
  <(adn P) patient {<(adn P) ssn ‘666’> <(adn P) addr A>}>@s3

```

The three answer objects received from s_1 , s_2 and s_3 are

then merged into a single *patient* object.

3.3. The Security Aspect Specification

We use RBAC as the underlying security model of medication systems, and consider two security aspects specification. The RBAC model has the following components:

1. U, R, P and S (users, roles, permissions and sessions respectively), where P is the Cartesian product of operation OP and objects Obj ,
2. $PA \subseteq P \times R$, a many-to-many permission to role assignment relation,
3. $UA \subseteq U \times R$, a many-to-many user to role assignment relation,
4. $user, S \rightarrow U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session’s lifetime), and
5. $roles: S \rightarrow 2^R$, a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r | (user(s_i), r) \in UA\}$ (which can change with time) and session s_i has the permissions $\cup_{r \in roles(s_i)} \{p | (p, r) \in PA\}$.

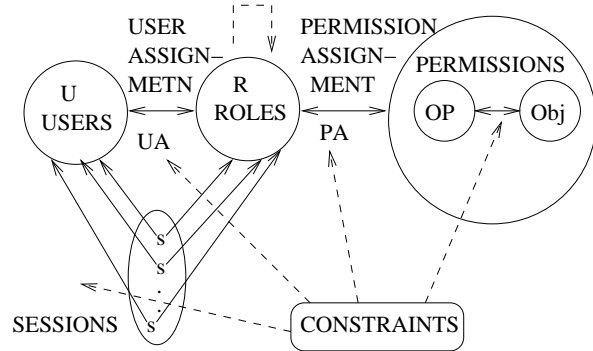


Figure 3. An RBAC model

In RBAC model permissions are associated with roles, and users are assigned roles based on their responsibilities and qualifications. In the mediation system, the security consideration permeates throughout the entire systems. It is possible to express security policies about the system that apply generically to a consideration, and then have the policies be applied through the system automatically. Two security aspects are identified as follows.

Aspect 1. check-role For the separation of duties principle, the same user can not be assigned to any two mutual exclusive roles simultaneously. Mutual exclusion in terms of user assignment specifies that one individual cannot be a member of both roles in exclusive sets. For instance, one

user can not play *patient* and *doctor* role in the same hospital simultaneously. The mediator specification in Section 3.2 only authentication the user and assign role to that user, but the mutual exclusive role checking is void.

Aspect *check_role()* prevents us from assigning the mutual exclusive roles r_1 and r_2 to the same user simultaneously. This aspect can be specified by the following predicate: $\forall u_1, u_2 \in U. (r_1 \in assigned_role(u_1) \wedge r_2 \in assigned_role(u_2) \rightarrow (u_1 \neq u_2))$, where *assigned_roles(u)* denotes the set of roles assigned to the user u . This aspect is from the constraints on the user assignment.

Aspect 2. check-view In the scenario that the user is allowed to retrieve data *name* and *xray* from source s_1 and s_2 by *ssn* respectively, but the tuple (*name*, *xray*) is sensitive global information. The mediator specifications described in section 3.2 failed to provide the global information filtering mechanism, although they provide the functionality to check the data permission against the data source by the predicate *check_permission()*. When the constraints on the local sources can not protect the information properly by its own, the global level security checking need be enforced into the mediator specifications. It can be used to automatically perform global security checking on sensitive global data retrieval.

Aspect *check_view()* filters the global sensitive data (*name*, *xray*) by checking the mediator views. *Check_view* aspect can be specified by the following predicate: $\forall v, name_0, xray_0, ssn_1, name_1, ssn_2, xray_2. (\neg echo(v, name_0, xray_0) \wedge (echo(v, ssn_1, name_1) \wedge echo(v, ssn_2, xray_2) \rightarrow ssn_1 \neq ssn_2))$, where predicate *echo(v, name₀, xray₀)* denotes that the view v can simultaneously feedbacks the attribute *name*'s value $name_0$ and *xray*'s value $xray_0$, and *echo(v, ssn₁, name₁)* denotes that the view v can simultaneously feedbacks the attribute *ssn*'s value ssn_1 and *name*'s value $name_1$. Similarly does *echo(v, ssn₂, xray₂)*. This aspect is from the constraints on the permission assignment.

3.4. Aspect Weaving

The purpose of aspect weaving is to process the components specification and the aspects specification, and to compose them properly into an integrated specification. Essential to aspect weaving is to specify the join points, where the functionality components and security aspects interact, and to define advices, which encode the appropriate behaviors at the join points.

The component specification language is Datalog and the join points are security-related predicates *role_assign* and

check_permission. The aspect specification language is based on first-order logic. Once a join point is met, there are several types of locations we can operate upon:

- Replacing the join point by well-defined procedure;
- Adding some codes before the join point;
- Adding some codes after the join point.

In our case, we only use the third option, i.e., adding *check_role* after *role_assign*, and adding *check_view* after *check_permission*. In addition to Datalog system, a theorem prover such PVS [14] can be used to automate the query process.

4. Conclusion

This paper proposes an aspect-oriented approach to enhancing security systems. RBAC serves as the security policy model and aspect-oriented approach is applied to isolate, compose and reuse the aspect specification. Two security aspects: *global information leaking* and *mutual exclusive roles* are identified and specified. And the identified weaver can integrates the aspect specification into the mediator specification.

The AOD method provides a rigorous way to identify notions in both functionality and security aspect specifications of mediation systems. The method supports reusable, and reliable design of secure mediator architectures. Security aspects express the essential issues of security requirements and security enforcement mechanisms, and are reusable across different systems. One can express security policies that are intended to be applied across a family of systems as aspects.

Several interesting research topics are: (1) How to properly partition the properties to be modeled by different aspect models, e.g. what information should be included in the base functionality model; (2) What is the dependency between the aspect models; (3) How to coordinate the security enforcement between mediation systems with local DBMSs; (4) How to automate the design process by efficient algorithms and tools.

References

- [1] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *SIGMOD Conference*, pages 137–148, 1996.
- [2] S. Adali and R. Emery. A uniform framework for integrating knowledge in heterogeneous knowledge systems. In *ICDE*, pages 513–520, 1995.

- [3] C. Altenschmidt and J. Biskup. Explicit representation of constrained schema mapping for mediated data integration. In *2nd Workshop on Databases in Networked Information Systems*, number 2544, Aizu, Japan, 2002.
- [4] Y. Arens, C. Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.
- [5] J. Biskup and D. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Source Information Systems archive*, 28(3):169–212, May 2003.
- [6] M. Carey and L. H. et al. Towards heterogeneous multimedia information systems: The Garlic approach. In *International Workshop on Research Issues in Data Engineering: Distributed Object Management*, Taipei, 1995.
- [7] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *The Eleventh International WWW Conference*, Hawaii, US, 2002.
- [8] R. K. Ege, L. Yang, Q. Kharma, and X. Ni. Three-layered mediator architecture based on dht (in press). In *International Symposium on Parallel Architecture, Algorithm and Networks (I-SPAN)*, Hong kong, 2004. IEEE press.
- [9] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y. Ng, D. Quass, and R. D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data Knowledge Engineering*, 31(3):227–251, 1999.
- [10] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
- [11] A. Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.
- [12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241*, pages 220–242. Springer-Verlag, 1997.
- [13] J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, 1994.
- [14] S. Owre, N. Shankar, J. Rushby, and D. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998.
- [15] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. ICDE Conf.*, pages 251–60, 1995.
- [16] M. T. Roth and P. Schwarz. Don’t scrap it, wrap it! a wrapper architecture for legacy sources. In *23th VLDB Conference*, pages 266–275, Athens, 1997.
- [17] R. Sandhu and E. Coyne. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- [18] C. Souza dos Santos, S. Abiteboul, and C. Delobel. Virtual schemas and bases. In *Proceedings of the International Conference on Extensive Data Base Technology (EDBT’94), Cambridge, UK*, pages 81–94. Springer-Verlag, Berlin, 1994.
- [19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, San Diego, August 2001.
- [20] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of disco. In *International Conference on Distributed Computing Systems*, pages 449–457, 1996.
- [21] V. Vassalos and Y. Papakonstantinou. Expressive capabilities description languages and query rewriting algorithms. *Journal of Logic Programming*, 43(1):75–122, 2000.
- [22] G. Wiederhold. Mediators in architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.
- [23] L. Yang and R. K. Ege. Modeling and verification of real-time mediation systems (in press). In *Advanced Simulation Technologies Conference*, Arlington, Virginia, April 2004.

Appendix

A. A Brief Introduction to Datalog

Some standard definitions and terminology on *Datalog* are introduced here.

An *atom* is a formula $p(t_1, \dots, t_n)$, where p is a predicate symbol and each t_i is a term (in the usual first-order logic sense). A term or an atom is said to be *ground* if it is variable free. A *fact* is a ground atom. A *database* is a finite set of facts. A *rule* is a formula written as $A \leftarrow B_1, \dots, B_m$ (in clausal form: $A \vee \neg B_1 \vee \dots \vee \neg B_m$), where A, B_1, \dots, B_m are atoms; A is called the *head* and B_1, \dots, B_m the *body* of the rule. A *logic program* is a finite set of rules together with a database. A *goal* is a formula written as $\leftarrow B_1, \dots, B_m$ (in clausal form: $\neg B_1 \vee \dots \vee \neg B_m$). A *query* is a finite set of rules together with a goal. All variables in rules and goals are implicitly universally quantified.

B. View

The notion of view is particularly important since it is used to consider the same object from various perspectives or with various precisions in its structure (e.g., for the integration of heterogeneous data). We need to specify complex restructuring operations. The view technology developed for object databases can be found in [18].

Declarative specification of a view: Following [18], a view can be defined by specifying the following: (i) how the object population is modified by hiding some objects and creating virtual objects; and how the relationship between objects is modified by hiding and adding edges between objects, or modifying edge labels.

This following example illustrates how to hide the person’s salary by exporting the view (*person*, “*high*”) from the stored database (*person*, *salary*) by the Datalog. (*person*, “*high*”) : $\neg(\text{person}, \text{salary}) \wedge \text{salary} > 100k$