Permissions meta-management architecture and BML tools for its implementation in the BlockSet project

Nikolay Alekseevich Kozyrev [a], Pavel Petrovich Keyno [a], Leonid Leonidovich Khoroshko [a], Alexander Yurievich Novikov [a] and Vladimir Mikhailovich Kvashnin [a]

[a] *Moscow Aviation Institute, Address, Moscow, Index, Russia*

Abstract

Web application development these days is a very common task but not always could be solved in an easy way. Especially when we are talking about a fully functional dynamic web application with an unlimited number of pages, as well as the ability to withstand the high performance in projects of this kind. The use of general-purpose programming languages for implementing the logic of the server side has a large entrance threshold, and traditional CMS and frameworks do not provide sufficient flexibility. The most correct solution in such situation would be to use DSL-like languages, which are easy to use and at the same time have extensive capabilities due to the fact that they are pre-oriented for solving domain-specific problems. However, at the moment there are very few ready-made solutions. Most known of them is WebDSL, but despite its purpose it has a number of disadvantages inherited from the implementation language of this project -- Java, such as, for example, little flexibility, high performance costs and the need to install a Java machine.

The development process was carried out using a relatively low-level general-purpose programming language C++ that resolved the problems faced by competitors, ensured high performance of the interpreter and cross-platform nature of the entire system.

The authors described in detail the chosen permissions meta-management tools of the BlockSet project, as well as the syntax of their application in the declarative highly abstract domain-specific language BML (BlockSet Modeling Language), using simple and accessible examples to demonstrate the concise, but expressive syntactic solutions.

The obtained results can significantly simplify the further process of administering a web resource and it's also one of the main features of the BlockSet project.

Thus, the BML language, coupled with the permissions system, provides flexible, powerful, and at the same time quite simple and understandable methods for organizing the necessary system configuration, which corresponds to the project paradigms of flexibility and low entrance threshold, and also significantly optimizes the development process.

Keywords 1

Metamodeling, permissions, user, group, DSL, domain-specific language, declarative, programming, language, web, BlockSet, BML.

1. Introduction

Web application development these days is a very common but not always easy task. Especially when we are talking about a fully functional dynamic web application with an unlimited number of pages, as well as the ability to withstand the high performance inherent to this kind of projects.

WCES 2021: Workshop on Control and Embedded Systems, May 01, 2021, Chennai, India.
EMAIL: pavel@mati.ru (Pavel Petrovich Keyno)
ORCID: 0000-0002-1033-0260 (Pavel Petrovich Keyno)

The use of general-purpose programming languages for implementing the logic of the server side has a large entrance threshold, and traditional CMS and frameworks do not provide sufficient flexibility. The most correct solution in such a situation would be to use DSL-like languages, which are easy to use and at the same time have extensive capabilities due to the fact that they are pre-oriented for solving domain-specific problems. However, at the moment there are very few ready-made solutions. When studying existing solutions, WebDSL turned out to be the closest to BlockSet in functionality, but it has a number of disadvantages inherited from the implementation language of this project -- Java, such as high-performance costs and the need to install a Java machine.

The BlockSet project compares favorably with competitors, solving the problems described above without loss of flexibility and performance and provides tools for implementing each of the stages of creating dynamic web applications. The most basic of such tools is the declarative, high-level, domain-specific language BML [3-5], based on semantically neutral XML. This language is used to describe the back-end logic of user requests, and the resource database, providing extensive opportunities for the implementation of this task, and its interpreter that is written in a relatively low-level C ++ language [6], which contributes to performance and ensures the cross-platform of this solution without unnecessary labor costs.

In addition to the tool for describing the logic of the server side, the BlockSet project also provides a flexible permissions meta-management system, it is important to provide the user with the opportunity to implement the system in such a way that for each user from whom an http request comes, the permissions for each information fragment in the database are strictly defined. This problem has already been raised in [1] and [2], but at a time it was not possible to find a complete and comprehensive solution to the problem. Obviously, the possibility of describing this system should also be present in the BML language, since the task of determining the rights of a specific user, the right to perform a specific operation with a specific piece of data is directly related to the logic of the database in general. When we are setting the task, it is worthwhile to clarify once again that the entire system, as well as the BML tools for its implementation, the design of which will be discussed in this article, are aimed not at managing the access rights of a web application, but at creating an environment in which it would be comfortable to implement and manage it, which is already a difficult and little-studied task, but in addition to this, it was also required to comply with the basic principles of the BlockSet project, such as flexibility sufficient to recreate approximately ~80% of the websites of Runet and with a low entrance threshold.

2. BlockSet Architecture

The main goal of BlockSet architecture is to provide simple, easy-to-use and limited number of entities. The idea is to make possible to work with it without providing additional attributes and properties for the entities. The developer will provide qualifying attributes later if it necessary. So, it’s strictly satisfied the requirements of declarative and domain-specific principle where we are already have some entity and we could work with it “from the box”. Thus, the developer who is just trying to learn something about BlockSet could achieve the milestones step-by-step by using additional attributes which makes the result project is flexible and more complex.

For a more understandable presentation, in the future, it is necessary to describe in general terms the main architecture of the BlockSet project and the entities it operates with. At first it is necessary to define the BlockSet in general. BlockSet is a web applications development toolkit. It provides a stack of technologies that can be collated to the common ones used everywhere for such purposes (figure 1).

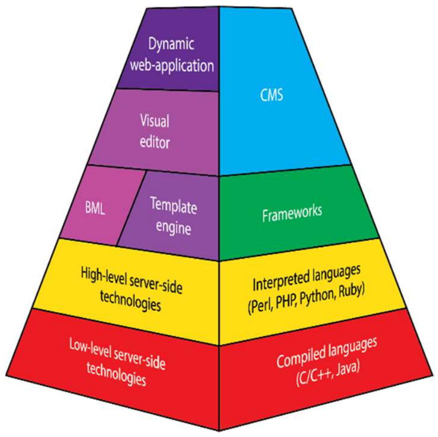

Dynamic web-application
CMS
Visual editor
BML
Template engine
Frameworks
High-level server-side technologies
Interpreted languages (Perl, PHP, Python, Ruby)
Low-level server-side technologies
Compiled languages (C/C++, Java)

Figure 1: Technologies Collation

2.1. Basic Definitions

The BlockSet toolkit operates with the following entities:

- *model*. There is always only one model per a project. It defines the general structure of the database and sets relationships between each other
- *location*. There are usually several locations per project. Each location describes the interaction logic between the database and request from the user within the framework of a specific URI of the resource

To accomplish their tasks, the model and locations operate with the following entities:

- *block*. A block is some atomic data fragment. String, number, time, email. Each block must be inside the set
- *set*. A set is a collection of blocks of various types, each of them is related to some abstract entity that it seeks to implement. May be contained within a model, location, or other set. The nesting of the sets determines the order of the selection.

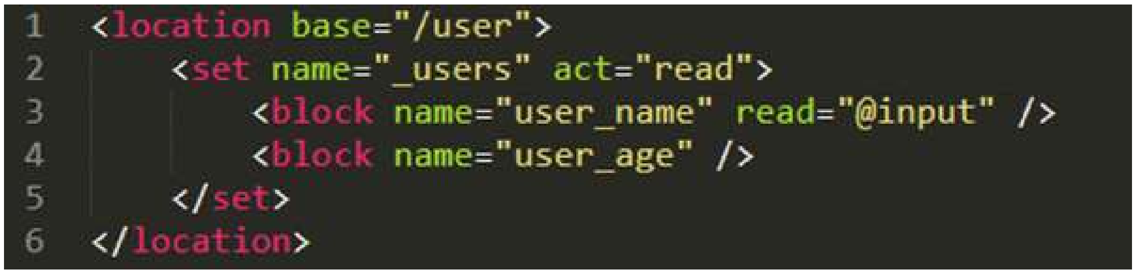
For example, the "_users" set might contain blocks such as "name", "age", or "_password". The BML language structure is described in more detail in other works [3, 4].

2.2. Flexible permissions meta-management system problem

In the common development of a dynamic web application, the obvious solution is often to write the logic of rights as part of the server side itself, that is, such code will be written in advance, taking into account the specifics of the application being developed, it will have the capabilities, the need for was described in advance. The difficulty of developing such a system for such a project as BlockSet is conditioned by the fact that requirements are not known in advance, and given the paradigm of the flexibility of this project, it is necessary to take into account all, or almost all, possible requirements for creating such systems and create tools for their implementation.

2.3. Requests handling

Permissions verification is performed on the server side, which complies with data security principles. At the entrance from the client, the server receives two blocks of information. First of all, this is a request for an operation that the user wants to perform with some data in the database. The data itself transmitted by the client is actually just the value of some variables that are used in the algorithm of actions with the database described using the BML language on the server side. For example, a username can be transmitted from a client, which will be used as a selection criterion for obtaining the age that we want to receive, but the transfer of a login in order to find a user by login will not lead to success, since the selection is described in the location exactly by the username, and not by his login (figure 2)



```
<location base="/user">
    <set name="_users" act="read">
        <block name="user_name" read="@input" />
        <block name="user_age" />
    </set>
</location>
```

Figure 2: Simple query example

In addition to the attribute values, the client also transmits the session key [3], which is generated on the server at the time of primary user authentication using a password, then returned to the user and serves as a unique session identification key. Using the key, the server performs a secondary authentication procedure for the initiator of the request and only after this stage the permissions check starts, which allows performing the requested operation, or rejecting it (figure 3).

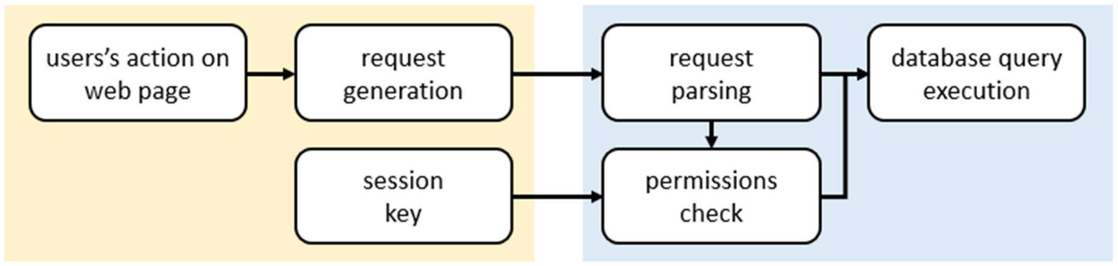


Figure 3: The client's request for an operation with data scheme

3. Permissions

This system is a specialized database configuration by different kind of permissions: for each cell (except for the primary key), in addition to the piece of information itself, users' permissions for CRUD operations (create, read, update, delete) [8] with cell. It should be noted that read and update can be defined for each attribute separately, when create and delete are defined for the entire row.

3.1. Groups and users

Almost any permissions system is based on such fundamental concepts as users and groups. Groups are needed to manipulate the permissions of several users at once, since it is not best practice to separately determine the permissions of each individual user, but such a possibility is still present in the calculation for cases when it is really necessary.

To store groups and users in the database, the corresponding system entities ***_groups*** and ***_users*** are used, as well as the connecting entity ***_groups $ _users***, which describes the belonging of each user to one or another group. It is important to mention that each group has such a parameter as priority. Of all the groups in which the user is a member, group permissions with the highest priority will be applied first to determine the actual rights. Groups are of three types: system, static and scoped.

3.1.1. System groups

The logic of system group processing is predetermined. The functionality of adding new groups of this type is not provided, since their logic is closely related to web entities that are not defined at the BML level. These groups include:

- ***all*** applies to all users, but has the lowest priority
- ***users*** applies to all authenticated and authorized users
- ***creator*** applies to the user initiating the creation of the current row in the database

3.1.2. Static groups

A user's membership in such a group is determined only from the strictly specified presence of a user in this group in a ***_users$_groups*** table.

3.1.3. Scoped groups

Membership in a group of this type is determined separately for each scope. A good example of implementing a group of this type is the obvious implementation of the ***forum_moderator*** group. Implementation of such a group will provide to its members the opportunity for special operations in a whole subtree of topics and messages of this special one forum (figure 4).

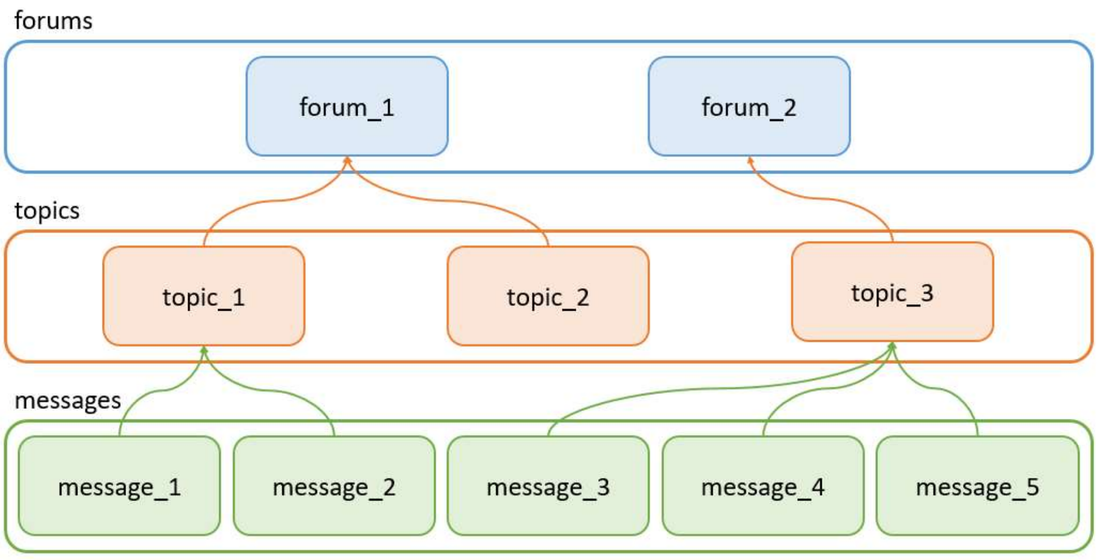


Figure 4: Scope groups example

3.2. Permissions general architecture and storage

The differentiation of permissions by operation type is carried out between four main generally accepted operations: create, read, update, delete, abbreviated as CRUD [4] [5], as mentioned above. To store the permissions in the database, special-formatted strings are used to set the permissions for one user or group

Access rights are a priori associated with data, and data are stored in the database, therefore, within the framework of this article, various types of rights will be described based on the database entities to which they refer. To solve the problem of flexibility, stored permissions are located directly in the database itself in such a way that each piece of information can be associated with a separate set of permissions. To store such a complex structure, the PostgreSQL DBMS, which is gaining popularity in recent years, is used, which, by means of composite data types [6], avoids high performance costs when storing structures in the form of JSON [7], as it was implemented in other DBMS, like MySQL.

It also avoids the cumbersome structure of the database on creating additional tables. Thus, all permissions to all attributes in the database are stored in a separate column and do not litter the table.

3.3. Permissions types

The BlockSet toolkit provides two types of permissions. These include local and remote. Local permissions are stored in the database per each entity, while the remote ones resolved by referencing to the permission-interface blocks in the same on another database table. They can be very useful to grant some set of users the availability to determine users activities permissions on something. (figure 5)

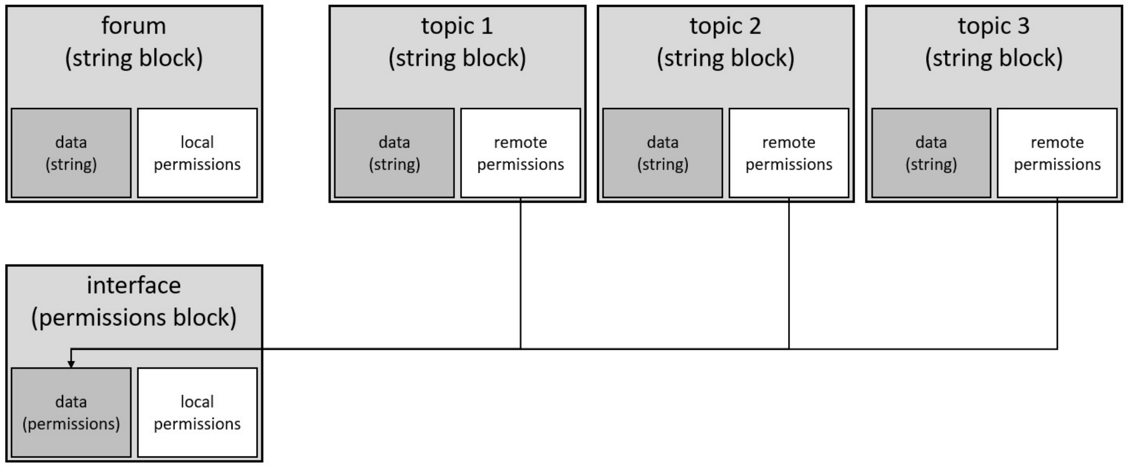


Figure 5: Interface block example

3.4. Attributes permissions

Local permissions for attributes are a set of permissions for a user or group in the following format: ***unit:ru***, where ***unit*** is a user ID or group name, ***r*** is the ability of the specified ***unit*** (user or group) to read the attribute, and ***u*** is the ability to change it. ***r*** and ***u*** can take the values ***y***, ***n*** or **-**, which respectively declares allow, deny or inheritance for corresponding operation. For example, the line ***@users: nn*** will prevent the registered users group from reading and changing the selected attribute, and the line ***355: y-*** will allow the user with ID 355 to read the data, but the ability to update will be inherited from the lower priority rule.

Remote rights for attributes are a collection of links. Each link must point to an interface cell, from which static rights will be taken, which will be applied to the attribute data. This links are not stored in the database in order to maintain sets structure consistency.

3.5. Tables permissions

Local permissions for tables are a set of permissions for a user or group. The logic of their work corresponds to the logic of work of local permissions for attributes, but unlike them, the rights here use a full-fledged crud and have the format ***unit:crud***. ***r*** defines the ability to select rows, ***u*** defines the ability to modify them, when ***c*** and ***d*** define the ability to create and delete, respectively. Such permissions are stored in a special table, where a row is provided for each other table, local rights are required for.

Remote permissions for attributes are a collection of links. The logic of their work is completely similar to the logic of the remote permissions for attributes, however, the storage of such rights is implemented in the same way as the storage of remote permissions for attributes.

3.6. Relations permissions

Local and remote permissions for one-to-many and many-to-one relationships are implemented in the same way as for attributes. But in this case, the permission target is the table's foreign key. Only have read and update permissions. In the case of a many-to-many relationship, the programming logic of such rights is a special case of rights for tables, since the rights to join tables are stored alongside the permissions to other tables.

4. BML toolset

The permissions for a particular block are stored directly in the database, as described above. The task of the BML language in this case is precisely meta-management, that is, the providing of environment for convenient permissions system implementing and managing. We are talking about default permissions, i.e. permissions that should be automatically set for a particular block or set during its creation, as well as on the generation of interface blocks. For example, a popular solution would be to declare the ***posts_interface*** block in the ***_users*** set and provide the appropriate user with the necessary permissions to edit this block, as well as assign default rights to all posts, which will be represented by links to the block user interface on whose page it is located. Thus, the defined system will allow the user to edit the permissions for posts on his profile page.

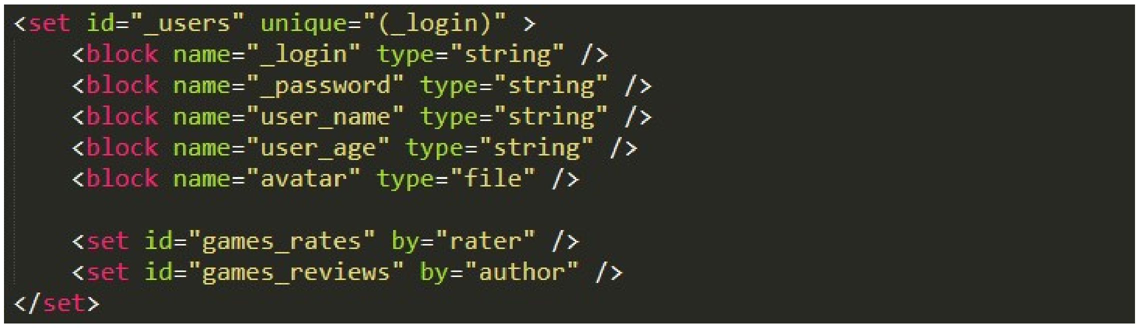
In the example above, there are seven main BML permissions meta-management tools that must be implemented:

- Interface block definition tool
- Attribute default local permissions definition tool
- Attribute remote permissions definition tool
- Table default local permissions definition tool
- Table remote permissions definition tool
- Relation default local permissions definition tool
- Relation remote permissions definition tool

Also, for the functioning of local permissions and convenient manipulation of them, it should be possible to create scoped groups and indicate which sets belong to them. Thus, to the list of required funds are added:

- Scoped groups definition tool
- Scope membership definition tool

For clarity, we will consider tools using the example of a set of users ***_users***, which has several blocks and belongs to the sets ***games_rates*** and ***games_reviews*** with an one-to-many relationship (figure 6)

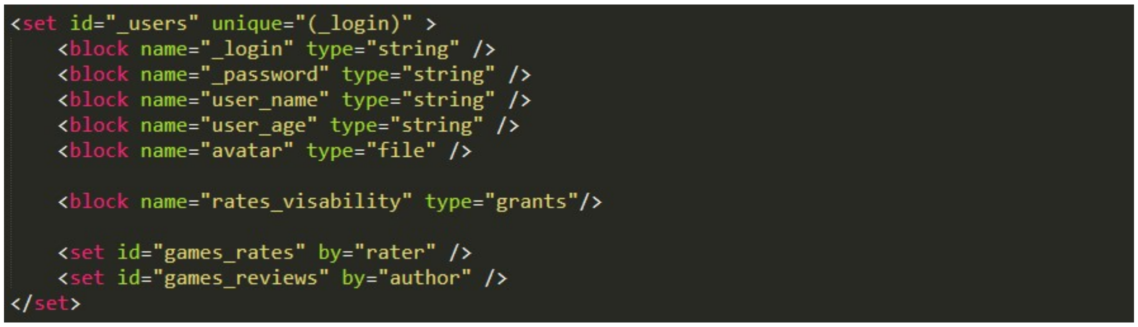


```
<set id="_users" unique="(_login)" >
    <block name="_login" type="string" />
    <block name="_password" type="string" />
    <block name="user_name" type="string" />
    <block name="user_age" type="string" />
    <block name="avatar" type="file" />

    <set id="games_rates" by="rater" />
    <set id="games_reviews" by="author" />
</set>
```

Figure 6: users set example

4.1. Interface block definition tool

The tool for interface blocks definition is represented by a tool whose syntax is similar to the syntax for describing other blocks (Fig. 1). In the model, a block must be declared inside the required set. We must specify the desired block name as the value of the ***name*** attribute, and the block type ***grants*** as the value of the ***type*** attribute. If necessary, define the default imported permissions (those that will be generated upon creation). The ability to set the permissions to operations with the block itself, both local and remote, is also implemented, as well as for any other block, it will be discussed below (figure 7).

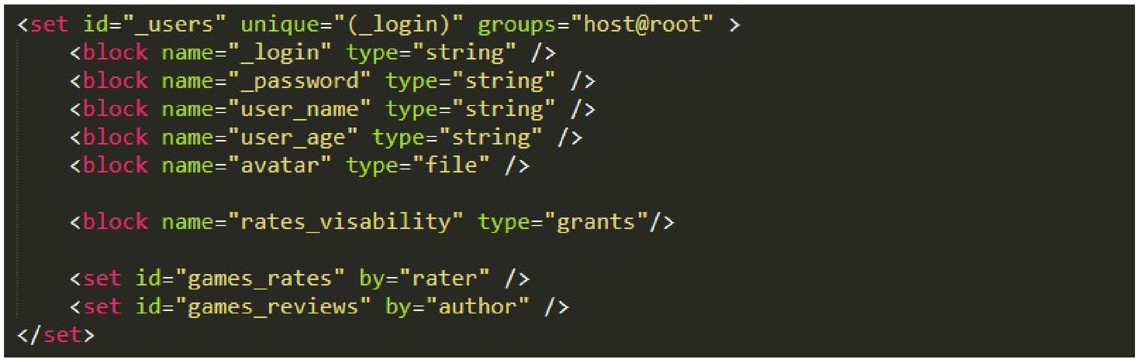


```
<set id="_users" unique="(_login)" >
    <block name="_login" type="string" />
    <block name="_password" type="string" />
    <block name="user_name" type="string" />
    <block name="user_age" type="string" />
    <block name="avatar" type="file" />

    <block name="rates_visability" type="grants"/>

    <set id="games_rates" by="rater" />
    <set id="games_reviews" by="author" />
</set>
```

Figure 7: Block-interface creation example

4.2. Scoped groups and its membership definition tool

Due to the fact that the functionality of these tools partially overlaps (for example, when creating a scoped group, you must specify a set that is the root of the scope), it was decided to implement them within one tool: to work with groups, the set attribute ***groups*** is used, within which, it is necessary to indicate the scoped groups it belongs to, separated by a space. In this case, it is also necessary to indicate the name of the relation with another set belonging to the specified group. If a set needs to be made root of a scope, the ***root*** keyword should be used instead of the relation name. Thus, the ***host@root*** entry in the ***groups*** attribute of the ***_users*** set will create the host scoped group (the profile owner) and make the ***_users*** set root within its scope, and the ***host@rater*** entry in the ***groups*** attribute of the ***rates*** will indicate that the ***rates*** set is a part of the ***host*** scope and is attached via the ***rater*** relation (figure 8).

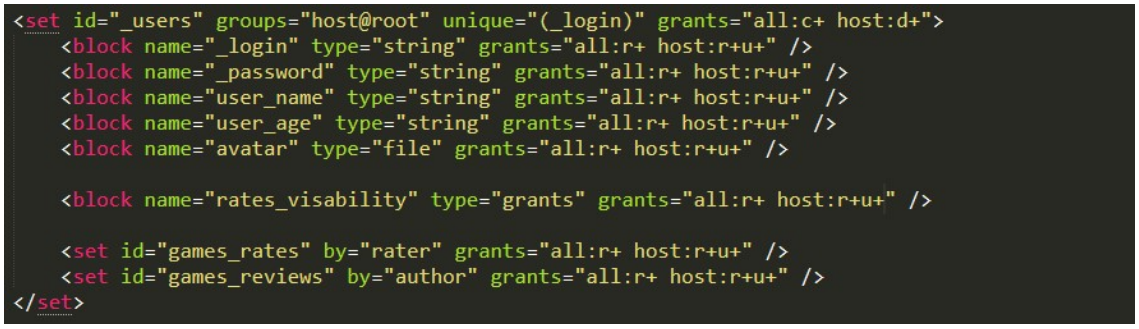


```
<set id="_users" unique="(_login)" groups="host@root" >
    <block name="_login" type="string" />
    <block name="_password" type="string" />
    <block name="user_name" type="string" />
    <block name="user_age" type="string" />
    <block name="avatar" type="file" />

    <block name="rates_visability" type="grants"/>

    <set id="games_rates" by="rater" />
    <set id="games_reviews" by="author" />
</set>
```

Figure 8: Scope group creation example

4.3. Local permissions definition tool

The definition means for local permissions for attributes, tables and relations is presented in a laconic and convenient tool: for the necessary block, set or relation, it is enough to define the value of the ***grants*** attribute, which must be represented by a line that includes local permissions listed in the format for a set or block separated by a space respectively. The format is as follows: first, the name of the group, the permissions are defined to, specialized, and then the necessary permissions are indicated after the colon. Each permission is a letter (c, r, u, or d) and the permission value itself is "+" for allow permission, "-" for deny, and "~" for inheritance. Moreover, for blocks and links, only the letters r and u are allowed, and for sets - all four (figure 9).

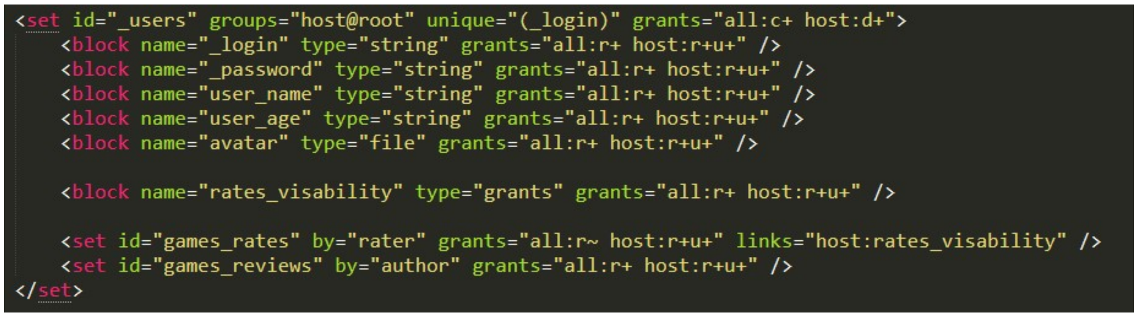


```
<set id="_users" groups="host@root" unique="(_login)" grants="all:c+ host:d+">
    <block name="_login" type="string" grants="all:r+ host:r+u+" />
    <block name="_password" type="string" grants="all:r+ host:r+u+" />
    <block name="user_name" type="string" grants="all:r+ host:r+u+" />
    <block name="user_age" type="string" grants="all:r+ host:r+u+" />
    <block name="avatar" type="file" grants="all:r+ host:r+u+" />

    <block name="rates_visability" type="grants" grants="all:r+ host:r+u+" />

    <set id="games_rates" by="rater" grants="all:r+ host:r+u+" />
    <set id="games_reviews" by="author" grants="all:r+ host:r+u+" />
</set>
```

Figure 9: Static grants creation example

4.4. Remote permissions definition tool

The means for defining remote permissions for attributes, tables and relations are represented by a similar tool, but instead of the ***grants*** attribute, you need to work with the ***links*** attribute in this case, in its value you should also specify links, separated by a space. Each link works only within the specified scope. This restriction allows us to significantly simplify the syntax since there is no need to specify the path to the interface and all that remains is to specify the scope name and interface name due to the fact that the hierarchy of sets within the scope has already been established using the tool for specifying scoped groups. Thus, each link is a ***zone-interface*** pair separated by a colon. When parsed and stored in sets structure, links have a slightly more complex syntax and structure, allowing, on their own, to carry out independent transitions without unnecessary operations. Remote permissions have the lowest priority, so only those permissions that have received inheritance status at the lowest priority of local permissions will be calculated based on them. Thanks to such a hierarchy, it becomes possible to use local permissions as a mask for remote ones, limiting the ability of the user to regulate rights (figure 10).



```
<set id="_users" groups="host@root" unique="(_login)" grants="all:c+ host:d+">
    <block name="_login" type="string" grants="all:r+ host:r+u+" />
    <block name="_password" type="string" grants="all:r+ host:r+u+" />
    <block name="user_name" type="string" grants="all:r+ host:r+u+" />
    <block name="user_age" type="string" grants="all:r+ host:r+u+" />
    <block name="avatar" type="file" grants="all:r+ host:r+u+" />

    <block name="rates_visability" type="grants" grants="all:r+ host:r+u+" />

    <set id="games_rates" by="rater" grants="all:r~ host:r+u+" links="host:rates_visability" />
    <set id="games_reviews" by="author" grants="all:r+ host:r+u+" />
</set>
```

Figure 10: Dynamic grants definition example

5. Conclusion

The design of the permissions metamodeling is the essential part of any project based on domain-specific and model-driven principles which processing a data based on different criteria. It's important to provide easy-to-use and at the same time flexible abstract system of permissions handling. As the whole project BlockSet permissions system provides high-abstract and declarative syntax implemented in BML language.

Ensuring information security is an integral part of the development and administration of a web application, for the organization of which it is necessary to spend a significant amount of time and effort, as well as to have the necessary knowledge and development experience using the appropriate low-abstract general-purpose programming languages.

Thanks to the work done for the BML language, it was possible to design tools for meta-management of access rights, that is, tools with which the possibility of creating a system is realized, within which it will be possible to configure access rights for a specific web application. During the design and development, it was possible to comply with the principles of the BlockSet project about flexibility and a low threshold of entry.

6. References

[1] Visser E. WebDSL. A Case Study in Domain-Specific Language. Berlin, Heidelberg. Springer 2008

[2] Keyno P.P.. Predposylki formirovanija novoj metodologii razrabotki web-uzlov BlockSet i declarativnogo jazyka BML [Prerequisites for the formation of a new methodology for the development of BlockSet web sites and the declarative languageBML] // Sovremennye informatsionnye tehnologii i IT-obrazovanie [Modern information technology and IT education]. 2015. vol. 11. № 2. pp. 78-84.

[3] Keyno P. P, Siluyanov A. V. Design and implementation of a declarative web-interface modeling lan-guage interpreter on a high-perfomance distributed systems. Prikladnaya informatika — Journal of Applied Informatics, 2015, vol. 10, no. 1, pp. 5-20

[4] Hanus M., Koschinicke S. An ER-based framework for declarative web programming // Practical Aspects of Declarative Languages. 2010. pp. 201-216

[5] P. Keyno, N. Kozyrev, V. Kvashnin, A. Novikov. Permission system and its management organization in BlockSet project. Prikladnaya informatika — Journal of Applied Informatics, 2019, vol. 14, no. 3 (81), pp. 66 — 73 (in Russian). DOI: 10.24411/ 1993-8314-2019-10016

[6] Bjarne Stroustrup. C++ programming language

[7] Truica C. O. et al. Performance evaluation for CRUD operations in asynchronously replicated document oriented database // Control Systems and Computer Science (CSCS), 2015 20th International Conference on. IEEE, 2015. pp. 191-196..

[8] *Cadavid J. J. et al. A Domain Specific Language to Generate Web Applications//CIbSE. 2009. Pp. 139-144.*

[9] Borovskij I.G., Shelmina E.A.. Sravnitelnyj analiz nasctol'nyh i client-servernyh subd [Comparative analisis of desktop and client-server dbms]// From TUSUR's digest, Vol. 20 №4. P. 92 – 94

[10] The PostgreSQL Global Development Group. PostgreSQL documentation: Composites. URL: http://www.postgresql.org/docs/10/rowtypes.html (access date 24.04.2019)

[11] The PostgreSQL Global Development Group. PostgreSQL documentation: Arrays. URL: http://www.postgresql.org/docs/10/arrays.html (access data 24.04.2019)

[12] Nicolas Seriot. Parsing JSON is a Minefield. URL: htpp://http://seriot.ch/parsing_json.php (access date 24.04.2019).