STUDIA INFORMATICA 2003
Volume 24 Number 3 (55)

Piotr CZEKALSKI
Silesian University of Technology, Institute of Computer Science

AUTOMATED USER'S ROLE GENERATION FROM UML USE CASE DIAGRAMS

Summary. The paper describes an idea of automated generation of user system roles (groups) using UML Use Case Diagrams, applied to Relational Database Management Systems (RDBMSs)-based applications. The paper shows the usefulness of this method on sample Web-based application and contains some useful hints for developers implementing this idea. A proposal for relevant UML extension is also discussed, along with several implementation considerations.

Keywords: UML, automation, RDBMS, UCD, role, user.

AUTOMATYCZNE GENEROWANIE SZABLONÓW PRAW UŻYTKOWNIKÓW W OPARCIU O DIAGRAMY PRZYPADKÓW UŻYCIA UML

Streszczenie. Artykuł opisuje propozycję automatycznego generowania szablonów praw użytkowników (ról/grup) na podstawie diagramów przypadków użycia UML. Artykuł wskazuje użyteczność tej propozycji na podstawie przykładowej aplikacji bazodanowej opartej na WEB. W treści zawarte są wskazówki dotyczące implementacji automatu generującego. Propozycja obejmuje rozszerzenie języka UML.

Słowa kluczowe: UML, automatyzacja, RDBMS, UCD, rola, użytkownik.

1. UML – a first aid kit for system designers

UML (Unified Modeling Language) is a worldwide standard for software project representation. UML is a descendant of object oriented design and specifications methods used in 80's and 90's, generalizing and unifying Booch's, Rumbaugh's and Jacobson's methods [6]. Nowadays, most software systems use UML as a project description language

and as an easy way to communicate between customers, project managers, developers and other players in the design and implementation processes. UML contains several types of diagrams representing various aspects of the project. One of them is the Use Case Diagram, which represents typical scenarios involving users and software systems.

1.1. Use Case Diagram – does the system do what expected?

Use Case Diagram (UCD) is frequently used for gathering system requirements and is a very useful and powerful tool for representing these requirements to the customers (both ordinary users and experts). Therefore, it is one of the first and most frequently used diagrams in the process of system design and implementation (see [6], [7]). UCD contains actors, which represent users (or groups of users) or other parts in the system such as databases and so on. UCD also contains Use Cases of the system, representing some operations performed by actors. A sample UCD (see Fig. 1.) represents a system implementing dictionary operations with two kinds of actors (basic - passive operator only reading data; and advanced - active operator also editing data). UCD may include relations between use cases (include, extend and generalize). For more information about UCD see [5] and [6].

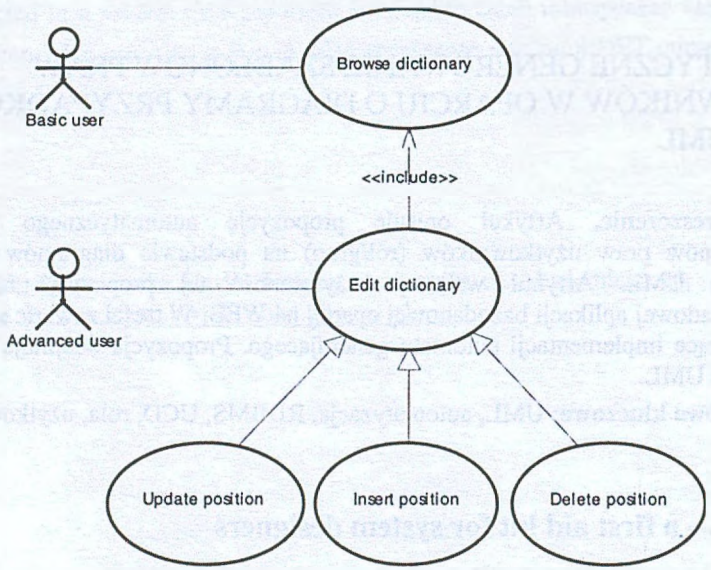

Fig. 1. Use Case Diagram – sample dictionary browser / editor
Rys. 1. Diagram przypadków użycia – przykładowa przeglądarka i edytor danych

2. Security systems

One of the most important features of all applications, particularly those working on database structures is a security system. Such a system defines rules, determining who can perform an operation and who can not. Most RDBMSs contain many user rights, both built-in and declared by developers, which can be grants, revokes and so on to the user.

Applications using databases can perform their internal security system checks or use built-in one (both solutions have advantages and disadvantages). An example system based on RDBMS, containing 900 database tables and 2000 users may have about 3000 elementary user rights that administrator must associate correctly to each user. To simplify administrative tasks, special security groups (roles) are being defined. The roles contain single security rights (grants, revokes) and each user is associated to some roles.

2.1. Security roles

While defining groups of user rights needs a lot of manual effort (which is usually developer's or installer's duty), using them accelerates administrative tasks efficiently. Such a group of rights is usually related to the position within business structure, as described by the role. The other advantage is flexibility in case of changes e.g. when a group of users is given new business tasks to perform, which empowers them to use new features of the system or when a user changes position within business structure achieving new responsibilities. The first case is a situation that usually entails adding right(s) to the group, initially deleting all granted rights to the users and then adding the new rights to them. The second case simply moves the user from one group to another. Concluding from above, the most time consuming task is preparing correct groups of user rights, based on business or other system analysis.

3. UML and automation

Can UML help to simplify the problem described in section 2? A good solution should provide the most of gain with the least effort. Automated code generation, based on UML diagrams is already implemented in various ranges of development tools, both commercial ([8], [9]) and none commercial. Automated generation brings many advantages to the system development process. Even such complicated tasks like automated performance estimation of projected systems can be done this way (see [1], [2] and [3]). Complex description of the tool implementing automated generation of simulation programs from UML diagrams can be found in [1] and [4].

3.1. Security administration

Most designs of complicated systems have UCD from the early stage of their development process. A business level UCD shows relationship between groups of users (like managers, secretaries, administrators, etc.) and the typical task that they perform (the Use Cases).

It is important to remember that a simple Use Case corresponds to a single transaction in terms of relational database model [7]. Such postulate finally points each Use Case to define a set of user rights, corresponding to the necessary "grants" and "denies" to the tables involved in the task. In case of a security system that operates at the application level, Use Case usually corresponds to a single "security key", which means a single check on the user right "on entry" of the secured process. If the security system uses database-integrated security, a single Use Case corresponds to a set of rights necessary to perform transaction. Both cases can be considered as a subject for automated roles generation based on UML UCD.

3.2. UML-based, automated generation assumptions

Automated generation of the user groups (roles) needs some extra guidelines and incorporates extensions to UCD. A Use Case representing a single transaction must provide information on granted (or denied) rights and the related object. Whenever a database integrated security system is being used, an object usually represents a table or view. In the case of integrated application level security system, the object is usually the security system itself. An Actor representing a role needs to provide extra information about the role name, and points to the user rights (Use Cases) that it should have. UCD may contain generalizations and dependencies between Use Cases. In such a case there must be additional information differing those relations that extend the Use Case to a container of security rights or those that provide some additional security behavior. Relations between Actors can be omitted as long as the security system does not implement any groups (roles) hierarchy.

3.2.1. Actor extensions proposal

It is no necessary to extend Actor's notation as long as the Actor's name is equal to the role name it generates. Any extra information may be added as pure text within Actor's description property (which cannot be seen on UCD, however most UML design tools provide extra description pages feature, e.g. [8]).

3.2.2. Use Case extensions proposal

A Use Case needs to show if it acts as a passive container (consists only of other user rights of related Use Cases) or as an active player. This can be shown using "Stereotype" – a well-known UML feature. Containers shall be given "<<security container>>" stereotype, while Use Cases related to security grants or revokes shall be given "<<security grant>>" or "<<security revoke>>" stereotype.

For each security-related Use Case, two additional properties are needed:

- object name to which the security grant / revoke applies (e.g. table name),
- grant / revoke type (e.g. SELECT).

UML design tools offer various ways of binding those properties with Use Case, e.g. "Attribute" or "Specification". In the case of the "Attribute" property being used, there is a proposal to use following notation:

- security_object: <object_name>
- security_grant: <grant_type>

where "object_name" is a phrase identifying the security object - a subject of considered grant/revoke; and "grant_type" is a phrase or keyword specifying the type of security right. In the case of using "Specification" property there is a proposal of the same notation as that of the "Attribute" property. However, each note has to start on a new line. In both cases one usually cannot see those properties directly on UCD. Fig. 2. shows an example of UCD with all new notation elements. Note that those elements are usually hidden on standard UCD and the possibilities of their visualization depend on the selected UML design tool. In this example, they are shown within the UML "note" element.

One should notice that values are implementation-dependant and vary from one DBMS to another. More implementation considerations can be found within sections 3.3 and 4.

3.3. Implementation considerations

The resulting SQL script should be generated according to the following rules:

- each actor should generate one role/group,
- each Use Case having <<security grant>> or <<security revoke>> stereotype, which is associated to a selected actor should generate a user right that it represents, for the role identified by the actor,
- when the Use Case takes part in a generalization or dependency, it is important to incorporate the related user rights into the final SQL script associated to the considered actor, according to the following rules:
    - if the Use Case includes or extends another Use Case, it should generate all user rights that are generated by the included Use Case, also adding it's

own user rights if it has <<security grant>> or <<security revoke>> stereotypes;

- if the Use Case represents a generalization of other Use Cases, it should generate all user rights that are represented by the generalized Use Cases, as well as adding it's own user rights in the case of <<security grant>> or <<security revoke>> stereotypes;
- if the Use Case represents a generalization and the stereotype type is <<security container>>, it should generate only dependent (from generalized Use Cases) user rights but none by itself,

- the final set of user rights for the selected role should be generated distinctly from all generated user rights.

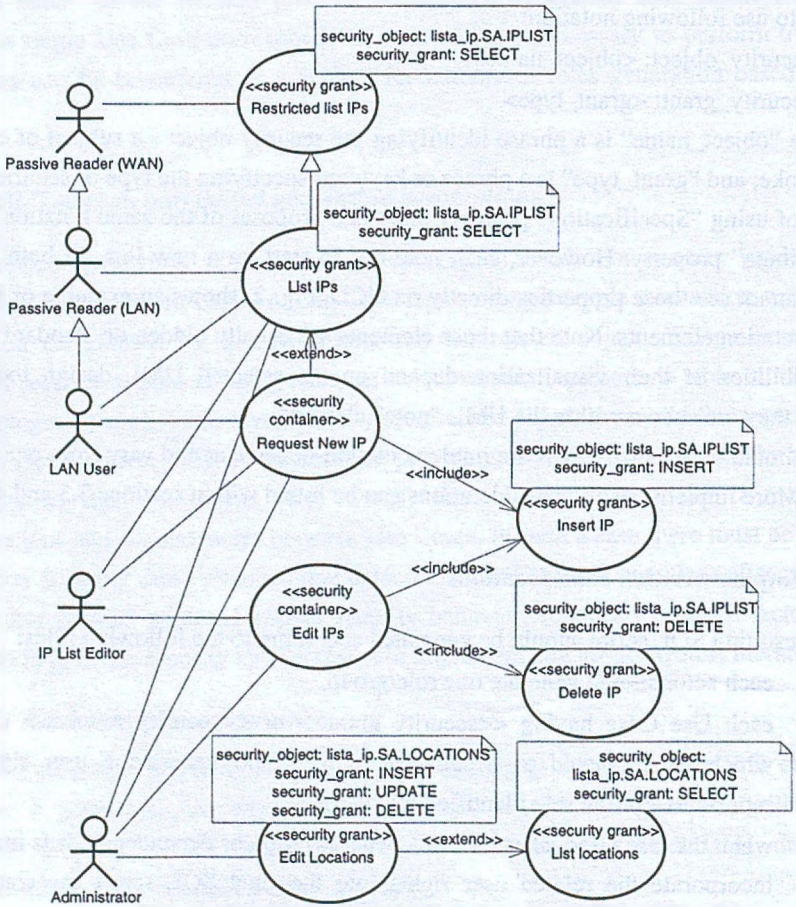

Fig. 2. Security oriented Use Case Diagram – sample IP recording application
Rys. 2. Diagram przypadków użycia zorientowany na zabezpieczenia – przykładowa aplikacja ewidencji adresów IP

3.4. Sample system

A sample UCD representing system that records IP addresses from a LAN (Local Area Network) can be seen on Fig. 2. The system contains Actors with different entitlements, several Use Cases, and relations between them (both "generalization" and "include / extend"). This UCD differs from the standard notation (see [5]), containing extensions described in paragraph 3.2.2. The resulting SQL script will contain 5 roles/groups, one for each actor: passive_reader_WAN (Wide Area Network), active_reader_LAN, LAN_user, IP_list_editor, administrator with the following user rights:

- passive_reader_WAN: select granted on IPLIST;
- active_reader_LAN: select granted on IPLIST;
- LAN_user: select and insert granted on IPLIST;
- IP_list_editor: select, insert, delete granted on IPLIST;
- administrator: select, insert and delete granted on IPLIST, select, insert and delete granted on LOCATIONS.

The above list shows the resulting set of rules and connected user rights that should be generated for the example system (see Fig. 2.) according to the algorithm of the UML-to-SQL script parser, explained briefly in section 3.3. More information about the parser's implementation can be found in section 4.

4. Implementation considerations

Implementation depends on the selected database platform. As most UML design tools offer a future for saving UCD into an XML (Extensible Markup Language) file, it is quite easy to prepare an XML parser, translating from XML data into an SQL script. Another solution is to prepare a dedicated application with graphical interface implementing UML UCD with extensions described in paragraph 3.2.1 and 3.2.2. An example of an application implementing graphical interface for UML diagrams design (written in Java), as well as a sample parser (written in Perl) can be found in [1]. Both tools can be easily adapted to solve considered problem of security related automated generation.

5. Conclusion

The described method simplifies administrative tasks related to the security system in large software environments both during installation and following maintenance.

REFERENCES

1. Arief L. B.: Ph.D. Thesis. A Framework for Supporting Automatic Simulation Generation from Design. University of Newcastle, Newcastle 2001.
2. Arief L. B., Speirs N. A.: Automated Generation of Distributed System Simulations from UML. 13th European Simulation Multiconference (ESM'99), Warszawa 1999, pp. 85-91.
3. Arief L. B., Speirs N. A.: Simulation Generation from UML Like Specifications. IAESTED Proc. International Conference on Applied Modeling and Simulation. Cairns 1999, pp. 384-388.
4. Arief L. B., Speirs N. A.: A UML tool for an Automatic Generation of Simulation Programs. ACM Proc. 2nd International Workshop on Software Performance, Ottawa 2000, pp. 71-76.
5. Booch G., Jacobson I., Rumbaugh J.: OMG Unified Modeling Language Specification Version 1.4. online at www.omg.org, 2001.
6. Fowler M., Scott K.: UML w kropelce. LTP, Warszawa 2002.
7. Muller R. J.: Bazy danych język UML w modelowaniu danych. MIKOM, Warszawa 2000
8. Sybase: PowerDesigner 9 Online Documentation. Sybase Inc., application online help, 2002.
9. Rational: Rational XDE Professional for Microsoft .Net Edition. Rational Software, online at www.rational.com, 2002.

Recenzent: Dr inż. Arkadiusz Sochan

Wpłynęło do Redakcji 25 marca 2003 r.

Streszczenie

Artykuł zawiera opis propozycji automatycznego generowania założeń dla grup / ról systemu użytkowników opartego na RDBMs na podstawie diagramów przypadków użycia UML. Zaproponowano stosowne rozszerzenie notacji UML. W treści znajdują się wskazówki dotyczące implementacji algorytmu generowania skryptów SQL, zawierających polecenia zakładania ról użytkowników i przydzielania im odpowiednich praw. Rozważania przedstawione w artykule nie dotyczą konkretnej platformy systemowej, stanowiąc elastyczną ideę, która może być implementowana w zależności od wybranego rozwiązania software'owego. W artykule zaproponowano dwie metody implementacji: generowanie

skryptów na podstawie dedykowanej aplikacji z interfejsem graficznym oraz generowanie w oparciu o translator kodu XML.

Adres

Piotr CZEKALSKI: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, bella@boy.iinf.polsl.gliwice.pl