

A Tool Supporting End-User Development of Access Control in Web Applications

Loredana Caruccio^{*,§}, Vincenzo Deufemia^{*,¶}, Christopher D'Souza^{†,‡,||},
Athula Ginige^{‡,***} and Giuseppe Polese^{*,††}

^{*}*University of Salerno, Via Giovanni Paolo II, 132
84084 Fisciano(SA), Italy*

[†]*Australian Catholic University, 40 Edward Street
North Sydney, NSW 2060, Australia*

[‡]*University of Western Sydney, Locked Bag 1797
Penrith, NSW 2751, Australia*

[§]*lcaruccio@unisa.it*

[¶]*deufemia@unisa.it*

^{||}*Christopher.D'Souza@acu.edu.au*

^{***}*a.ginige@uws.edu.au*

^{††}*gpolese@unisa.it*

End-user development (EUD) is drawing an increasing attention due to the necessity of users to frequently extend and personalize their applications. In particular, EUD in the context of Web (EUDWeb) is focusing on technologies capable of supporting development tasks that the end-user feels more complex. However, although the specification and implementation of access control is perceived as a particularly complex task, little efforts have been made to support it within current EUDWeb environments. Thus, in this paper we propose an EUDWeb framework and tool for the specification and the generation of web applications embedding access control mechanisms. We extended a previous mockup-based EUDWeb approach, by introducing visual assistance mechanisms enabling the specification of role-based access control policies, and their integration within the application logic. The usability of the proposed framework has been evaluated by means of a user study, in which we have shown that a group of heterogeneous end-users could proficiently use the proposed framework to develop meaningful web applications, some of which including access control functionalities.

Keywords: Human Computer Interaction; visual languages; web application modeling; access control; RBAC.

1. Introduction

Modern applications and mobile devices have contributed to increase the participation of users to the digital society [28]. People actively use applications, and often manifest the necessity to personalize them, but they cannot afford the costs of professional developers, nor they are willing to depend on them for the continuous

updates they need. For this reason, end-user development (EUD) has been boosted in the context of internet applications [18, 32], and many technologies are being proposed to empower end-users not only to autonomously personalize their applications, but also to develop several types of enterprise level internet applications [7, 22, 25].

EUDWeb technologies have focused on the support of development tasks that the end-user perceives as more complex [32], such as application control flow, state-less protocols, database management, access control, and so forth. In particular, concerning access control, while end-users are often able to understand and specify access control policies in their application domains, they might fail to embed them within a web application (Webapp) [32]. This is not a minor issue, since access control has long been identified as a necessary feature in applications, especially Webapps, due to the fact that they are remotely invoked by numerous users, by means of many different devices and software platforms, hence there are many more risks of malicious attacks.

In the literature, there are several proposals to simplify the specification of access control policies through visual languages, and for several types of software applications, including legacy ones [14]. However, although these proposals employ user-friendly notations, they have been mainly exploited in the context of professional programming activities.

In this paper we propose an EUDWeb framework and tool for the specification and the generation of Webapps embedding access control mechanisms. We extended a previous mockup-based EUDWeb approach and tool [10], by introducing visual assistance mechanisms enabling the specification of role-based access control (RBAC) policies [13, 33], supporting their implementation and integration within the Webapp logic. The framework is composed of three main visual environments: MODE enables end-users create both the look-and-feel of the application and its behavior by exploiting mockups; RODE enables end-users specify the role hierarchy; VULCAN provides the visual language for defining control actions. A pioneer version of the framework has been described in [5]. With respect to such a version, in this work we have improved the look and feel of the VULCAN environment and have provided a formal description of its underlying visual language. Moreover, we have provided initial validation of the whole framework by means of a user study which involved eight heterogenous end-users. The results show that the framework enables end-users build meaningful Webapps, also those including access control functionalities, thanks to the intuitiveness of the metaphors underlying the proposed environments, which made it possible to accomplish sufficiently complex programming tasks.

The proposed framework provides a twofold contribution to EUD research. First, through our mockup-based EUDWeb approach we make it possible handling access control within Webapps in a user-friendly fashion. Secondly, our visual notation enables the end-user condition the logic of single actions, based on the role associated to the user executing them.

The structure of the paper is as follows. In Sec. 2 we briefly review approaches using mockups for Webapp development, and discuss visual notations for the

specification of access policies. In Sec. 3 we introduce the proposed EUDWeb framework for generating Webapps embedding RBAC policies. The MODE and VULCAN environments are presented in Secs. 4 and 5, respectively. The architecture of the generated Webapp is detailed in Sec. 6, whereas the usability evaluation of the proposed framework is provided in Sec. 7. Finally, conclusion and future works end the paper in Sec. 8.

2. Related Work

In this section, we first discuss visual EUD approaches for the generation of Webapps. Then, we survey recent metaphors proposed for the definition and management of access and security policies.

2.1. EUD for web applications

Although EUD approaches and tools have been defined for simplifying the development of desktop applications [4, 23, 24], with the advent of Webapps this research area has received a renovated interest.

Many EUDWeb approaches and tools were intended for mashup development support, which can be seen as a special case of EUDWeb where a Webapp is constructed by integrating existing ones. For instance, Nichols and Lau present a system allowing users to create a mobile version of a Website by composing pieces of contents encountered during a web navigation session [26]. Similarly, Toomim *et al.* enable end-users compose their Webapp by selecting sample data from Websites, and automatically generate user interface (UI) enhancements [37]. Vegemite is a tool providing end-users with direct manipulation and programming-by-demonstration paradigms to populate tables, by extracting information from several Websites [21].

Alternatively, EUD environments have been proposed to compose Webapps from scratch, by following a model-driven approach [29, 30]. In this context, most proposals are based on UI mockups, due to their intuitiveness and effectiveness in requirement gathering and validation. In particular, in [30] the requirements gathering mockups created with the commercial software Balsamiq [1] are manually enriched with tags conveying semantics to its elements. This enables the translation of the enriched specification into an abstract UI model that is in turn converted into the presentation and the navigation models of a Webapp. Finally, after a series of transformations, developers achieve their executable prototype. The mockup-based approaches presented in [31] and [9] merely generate code implementing the UI skeleton, while the rest of the application needs to be manually coded.

The visual modeling approach proposed in [10] represents both static and behavioural information of a Webapp in a visual model. In particular, the visual model enables the specification of the Webapp look-and-feel through mockups, and of the user interactions through links, annotations, and widget references. End-users are guided during the modeling process through a summary view, for managing the

design of complex applications, and a data model view, for improving the quality of the generated applications.

Mockup-based navigational diagrams were proposed in [3] to formalize requirement analysis, in order to reduce the gap between end-user developers and software developers. Such diagrams contain essential components related to user interactions and navigational information. The mockup-based diagram is a graph where the nodes are either web pages or business entities, whilst the edges represent transition events.

Alternatively, the layout and the behaviour of a Webapp can be specified by means of window/event diagrams (WED), which represent a combination of UI models and state charts [35]. The tool supporting the WED notation is capable of generating code for prototype validation.

To our knowledge, none of the existing EUDWeb approaches and tools have explicitly and effectively addressed the problem of including access control within Webapps.

2.2. Visual specification of access control

Different models have been proposed to support the definition of rules and constraints for guaranteeing secure access to resources handled within information systems. Among these, it is worth mentioning access control list [20], task-based authorization control (TBAC) [36], and the role-based access control (RBAC) model [13, 33]. To simplify their definition and management, several visual metaphors have been proposed. As an example, the Miró system [16] provides two visual languages for the specification of authorization policies: an instance language for specifying an access matrix and a constraint language to specify possible restrictions of the system on which access permissions should be enforced. Such languages are based on hierarchical graphs, which make them difficult for unexperienced users. The Language for Security Constraints on Objects (LaSCO) exploits an annotated constraint graph to let users visually specify authorization policies [17], but it lacks a metaphor-based paradigm to facilitate its use to unexperienced users.

The visual language hierarchy (VTBAC) and the visual security administrator (VISA) system enable the visual specification of access control policies based on the TBAC model [6].

An alternative graphical specification of RBAC policies uses a generalization of string grammars to nonlinear structures [19], together with graph transformation tools. The model driven security (MDS) approach integrates system models and security requirements within a CASE-tool [2], enabling the automatic generation of system architectures, including complete, configured, access control infrastructures.

The approach proposed in this paper is inspired by the suite of visual languages presented in [15], used for specifying RBAC policies and for implementing them in the XACML language. In particular, the role diagram is used to model roles and relations among them, the permission diagram allows an administrator to specify the

access policies to the available resources, the separation of duties diagram is used to specify constraints, e.g. the resources that cannot be employed by users who have played a given role, and finally, the role assignment diagram is used to assign users to roles.

3. EUD of Access Control Policies in Web Applications: A Mockup-Driven Approach

The RBAC methodology is based on three fundamental concepts: user, role, and action, and on the idea that the association between a user and an action is related to the concept of role, which should be assigned to each user [34]. In other words, the association between single users and actions should indirectly occur through the definition of a policy of permissions (representing the triple role, resource, and action, including the associated rule), which is defined according to a role; and the association of users to roles. RBAC also includes the concept of role hierarchy, which provides the possibility to define hierarchical relationships between roles, through which it is possible to organize authority and responsibility.

One of the most frequently used languages to define access control policies in the RBAC methodology is the eXtensible Access Control Markup Language (XACML), which has also become a standard in 2005 [27]. Since XACML is an XML based language, it provides the advantage of implementation independence. Moreover, it enables the definition of access policies at different levels of detail, including the composition of policies, and the resolution of conflicts. Nevertheless, XACML is not a simple to use language, which has motivated the development of higher level languages from which XACML code could be automatically generated [39], including visual languages for particularly unexperienced users [15]. However, such languages have been embedded within integrated development environments (IDEs), hence they have been targeted to programmers, whereas it would be useful having them available within EUD environments for Webapps. Thus, in what follows, we describe a framework and a tool embedding visual languages for access control within a mockup-based EUDWeb environment.

Figure 1 shows the process underlying the proposed framework, through which the end-user will build his/her Webapp by means of three main environments: the ROle Diagram Environment (RODE), the MOckup Development Environment (MODE), and the VisUal Language for Control ActioN definition (VULCAN) environment. In particular, the end-user can use RODE to accomplish the first step in the specification of RBAC policies, that is, to specify the role hierarchy. An example of role hierarchy is shown in Fig. 2, where the role icons have associated a label, representing the role name, and a unique color to increase the level of user awareness. Moreover, the hierarchy represents an inheritance chain, which simplifies the task of end-users when specifying permissions for a role, since those specified for ancestor roles are inherited. In the example of Fig. 2 the role **Admin** inherits permissions from **Employee**, which in turn will inherit permissions from **User**.

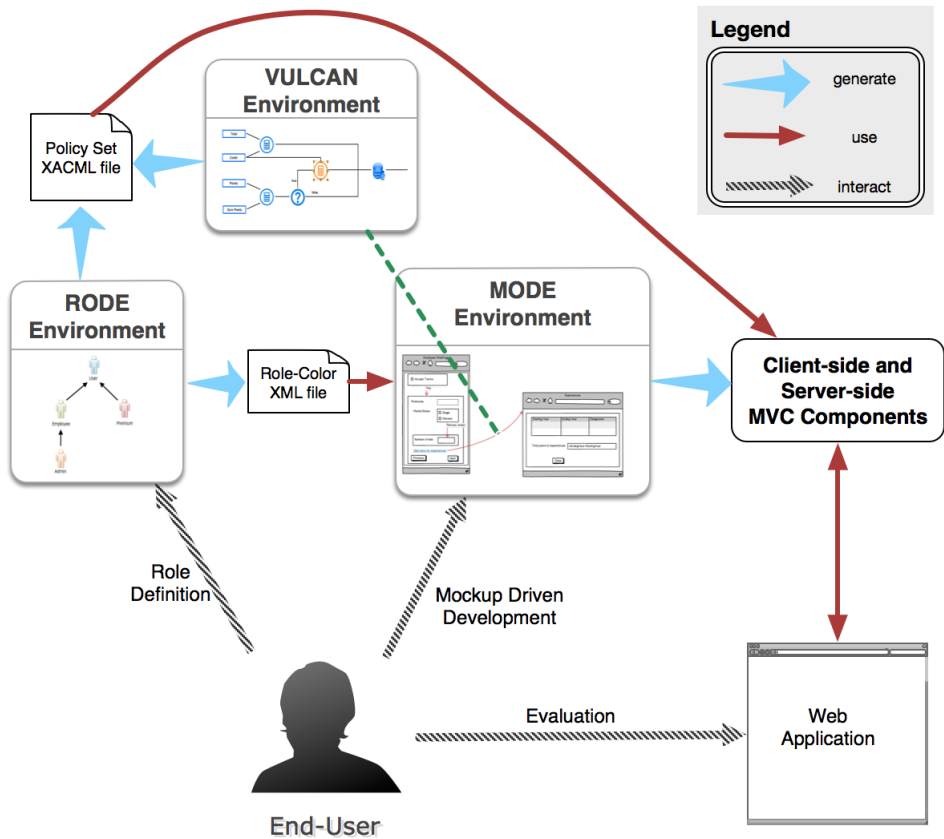


Fig. 1. The proposed EUDweb development process for generating Webapps embedding access control mechanisms.

Successively, the user specifies the whole Webapp by means of the MODE environment. In particular, in MODE the end-user can create the mockups of the web pages composing the Webapp, and enrich them with the specification of user interaction events, so enabling the automatic generation of the data model, the view, and the control logic of the whole Webapp [10]. After that, the end-user can specify the control flow of actions associated to user interaction events by means of the VULCAN environment. The latter also provides icon operators enabling the specification of access control policies associated to some of the specified actions, also exploiting the roles defined within RODE.

RODE also enables the automatic translation of the mappings between roles and colors into an XML file, which is exploited by MODE to highlight the roles authorized to access a given web (sub-)page. Moreover, both the specifications created with RODE and VULCAN are automatically translated into an XACML file (Policy Set), which is inquired by the generated Webapp in order to decide whether to grant or deny access to a requested resource.

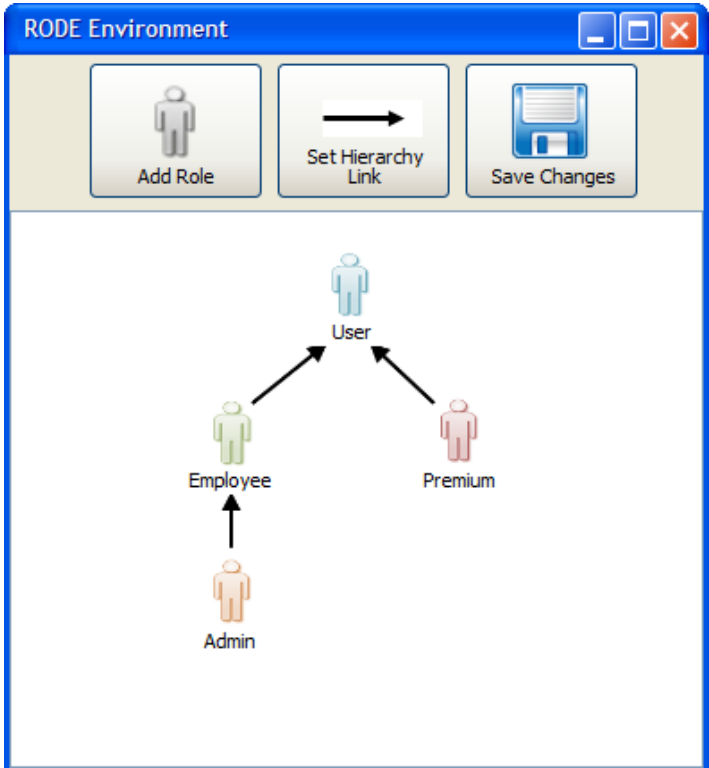


Fig. 2. The RODE environment with a role diagram.

In what follows, dedicated sections will provide details about the MODE and VULCAN environments. To this end, we will first introduce a running example to help illustrate concepts. In particular, we will refer to the specification of a Webapp for purchasing books and ebooks online. In this context, we will consider the following three roles: administrator, premium user, and user. The former has permissions to add new products to the online catalogue; the premium user can purchase both printed and online books, and is given the possibility to gain points for future discounts; finally, the user can only purchase printed books.

4. The MODE Environment

The MODE Environment extends the mockup-based EUDWeb approach proposed in [10] to embed the visual specification of actions associated to user interaction events, such as those related to the granting of access to given resources upon user requests.

In what follows, we first introduce the modeling process underlying MODE [10], and then we describe its extension to support the specification of control actions, including actions related to access control.

4.1. A mockup-driven modeling process for webapps

Typically, end-users specify the requirements of Webapps by sketching a set of UIs composing them. Based on this way of conceiving Webapps by end-users, the EUDWeb visual modeling approach underlying MODE starts with the creation of the mockup of the UI, which is enriched with user interaction information to automatically derive: the data model, the view, and the control logic of the Webapp [10], as shown in Fig. 3.

Figure 4 shows a portion of the mockup for the running example. In particular, the figure shows a mockup for the home page, which is colored according to the role *user* (see Fig. 2) to indicate that all the roles from user and higher can access it to enter login information. The orange colored mockups are only accessible to the role *administrator* to enter new books or search for existing ones, and to assign roles to users. The blue mockups are those accessible to all roles to search and purchase books, whereas the purple ones are only accessible to premium users, who are allowed to search and purchase ebooks, paying without entering credit card information.

The interaction behaviour is specified according to the Valverde and Pastor's dynamic model of the UI, which captures five essential aspects of behavioural information [38], embedded in the mockup as follows:

- the *navigational information* is visually modeled using a red arrow (for example, in Fig. 4, clicking on Add new eBook button on the Admin Home page causes the user navigate to the eBook Registration page);
- the *data request on demand* reaction is modeled using an assignment statement '=' in the widget that displays the requested data (for example, the expression '=Title' on the Book Search page issues a data request to the database storing the books),

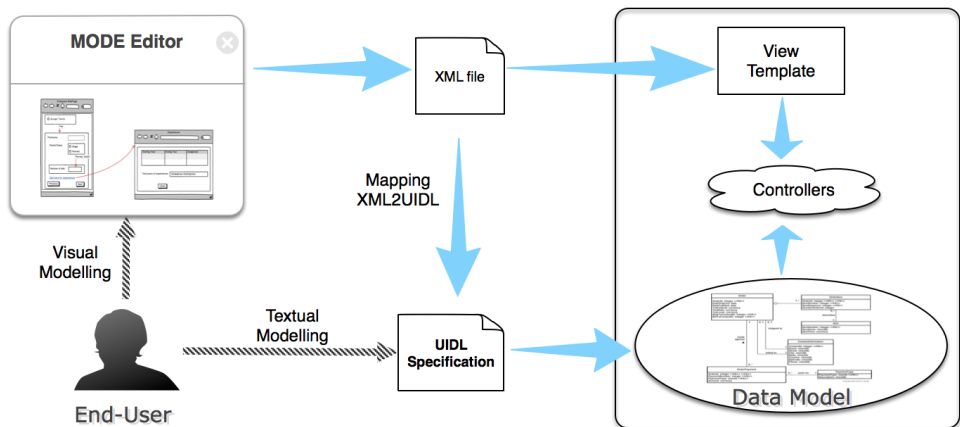


Fig. 3. The end-user modeling process proposed in [10].

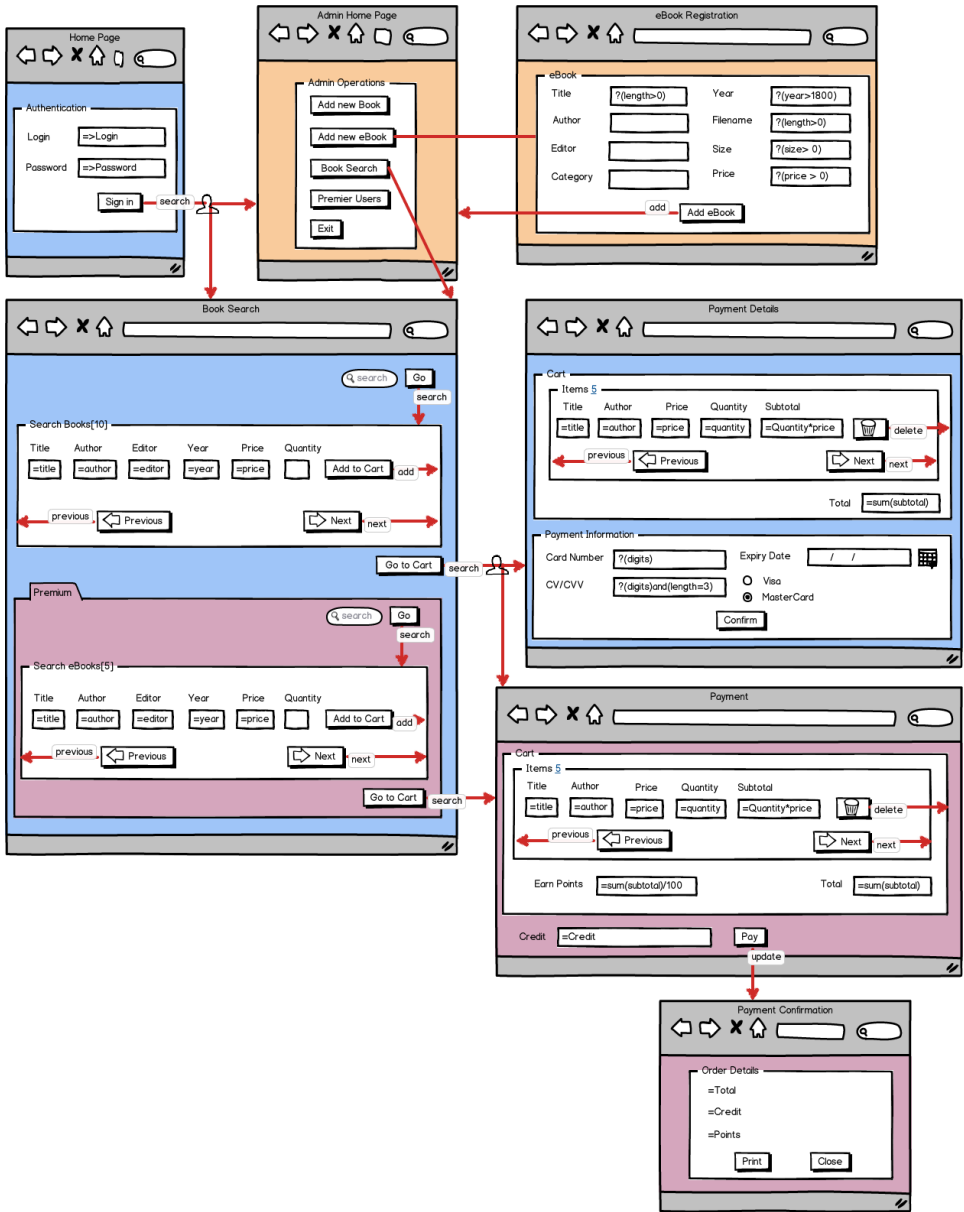


Fig. 4. Overview of UI mockup of a Webapp for purchasing books and ebooks.

- the *functional invocations* are visually captured by a red arrow along with an optional event and an action (for example, clicking the Submit button triggers the invocation of the local search method with login and password as arguments),

- the *input validation* is modeled by using the symbolic character ‘?’, followed by a validation rule specified in a suitable widget (for example, the **Card Number** widget has associated the validation rule `?(digits)` to constrain users input only digits), and
- the *property changes* are modeled as a red arrow between the source widget triggering the changes, and the target one undergoing the change.

The output of the visual specification is an XML document to be used for the automatic generation of the textual User Interface Description Language (UIDL) code, and a view template [10]. The UIDL code is a high-level textual specification equivalent to the visual one, and it can be used to define behaviours requiring complex computations, and to derive the data model. The view template defines concrete details, such as the actual position of widgets in a page, along with their styling features, in a platform independent fashion. Finally, the view templates and the data model are used to derive the control logic of the application.

4.2. Definition of access control actions

From the previous subsection we have seen that MODE enables end-users to develop their own Webapps through a WYSIWYG (What You See Is What You Get) paradigm. However, there might be some inner computations that the end-user is familiar with, which need not be specified within the UI. Thus, it is necessary to empower end-users with the possibility of specifying the behavior triggered by a specific UI event through metaphors based on their own enterprise processes.

In order to ensure a proper level of user-friendliness for such end-user process, we propose the visual language VULCAN (VisUal Language for Control ActioN definition) to define the application logic of actions triggered by UI events. In this context, particularly important can be the specification of actions concerning access control policies. In what follows, we detail how MODE supports the specification of this type of actions, and in particular, the metaphors used within mockups to recall roles and their permissions. The details of the VULCAN language will be discussed in Sec. 5.

4.2.1. Control flow specification

When specifying the control flow of an action, the end-user might want to condition it on the actor’s role. In the MODE environment this will be specified by placing a role icon on the arrow representing the action flow, immediately before a fork. Moreover, the background color of each target page will indicate the lowest role required to access it. As an example, in Fig. 4, the action to be executed upon clicking the **Sign in** button on the **Home Page** depends on the role of the actor. In particular, if the actor has the **Admin** role, then the orange colored **Admin Home Page** will be visualized, whereas actors with other roles will be directed towards a blue colored **Book Search** page.

The details on the role-based control flow definition can be defined through VULCAN, which enables the specification of aspects concerning the application logic underlying an action, as explained in the next section.

4.2.2. Visualization of the requested role within mockups






The role associated to an actor could also be taken into consideration to decide whether to visualize some containers of a page. This offers the possibility to define the same page for different roles, but some elements will only be visible to users with higher privileges. An example is reported in the Book Search page in Fig. 4, where the section of the search results for eBooks will only be visible to actors with at least the role Premium, even though the whole page is accessible to all users.

MODE also offers the end-user the possibility to visualize only the mockups accessible to a subset of the specified roles. This also contributes to provide support for debugging, which is perceived as one of the most complex activities for end-user in the EUDWeb process [32].

5. The VULCAN Visual Language






The icon dictionary of the VULCAN visual language is reported in Table 1. The first column depicts the icon for each icon operator, whereas the second column reports the name of the icon operator, a brief description, possibly the input and output operator, and finally, the configuration parameters.

Table 1. The VULCAN icon operators for specifying action rules.

| Icon | Summary |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | LINK: This operator links visual operators and indicates the execution flow in the action rule. |
|  | ITEM: This operator refers to the information defined in an HTML input element of the mockup or to the information contained in the input LINK operator. |
|  | CHECK: This operator defines a conditional statement determining the branch to be executed. INPUT OPERATORS: one or more LINKS PARAMETERS: a boolean expression on the values of the input LINKS OUTPUT: one or two LINKS, the one labeled <i>true</i> refers to the flow executed when the boolean expression is satisfied, whereas the other one (which is optional) is labelled <i>false</i> . |
|  | OPERATION: This operator enables operations on the input data. INPUT OPERATORS: one or more LINKS PARAMETERS: arithmetic expressions and/or functions on the input values OUTPUT: a set of values produced by the application of the arithmetic expressions and/or functions to the input data. |
|  | WEB SERVICE: This operator enables the invocation of web services. INPUT OPERATORS: one or more LINKS PARAMETERS: WSDL address, web service parameters OUTPUT: a LINK referring to the flow executed when the web service works correctly. |

(Continued)

Table 1. (Continued)

| Icon | Summary |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <p>ACCESS POLICY: This operator specifies a role-based policy for accessing a given resource. INPUT OPERATORS: one LINK whose associated values include the role PARAMETERS: list of policies specifying which roles can access to a given resource OUTPUT: one or more LINKS whose labels indicates the roles required to access the resource.</p> |
|  | <p>DB SEARCH: This operator specifies a database search operation. INPUT OPERATORS: one or more LINKS PARAMETERS: the data sources and the conditions on data OUTPUT: a LINK referring to the flow when the query is executed correctly.</p> |
|  | <p>DB INSERT: This operator specifies a database insert operation. INPUT OPERATORS: one or more LINKS PARAMETERS: the data sources OUTPUT: a LINK referring to the flow when the query is executed correctly.</p> |
|  | <p>DB UPDATE: This operator specifies a database update operation. INPUT OPERATORS: one or more LINKS PARAMETERS: the data sources and specific conditions for the application of the data update operation OUTPUT: a LINK referring to the flow when the query is executed correctly.</p> |
|  | <p>DB DELETE: This operator specifies a database delete operation. INPUT OPERATORS: one or more LINKS PARAMETERS: the data sources and specific conditions for the application of the data remove operation OUTPUT: a LINK referring to the flow when the query is executed correctly.</p> |

The parameters for each icon operator can be specified through a configuration panel, which appears in the UI when the operator is selected. As an example, Fig. 5 shows the screenshot of the panel for specifying the condition associated to a Check operator. The end-user can select the input values of the icon operator from the list **Items**, and the operation to be performed on them from the list **Operators**. S/he can optionally specify constant values for the defined condition.

Figure 6 shows another example of panel for the configuration of icon parameters. It enables the specification of properties of the Access Policy operator, by defining the roles that can access the resource, and possibly, the target resource. In particular, the resources are defined either in the MODE editor, as targets of action arrows, or in the panel of the Access Policy operator.

5.1. The VULCAN environment

The VULCAN environment provides an editor and a compiler for a visual language to be used for specifying control actions. It is activated from MODE upon double clicking on UI components. As shown in Fig. 7, the environment is composed of four main areas. The central area provides a workspace, in which the control flow of the action is specified. The top area provides the icon operators that the end-user can employ to define the application logic associated to the action. The left area provides

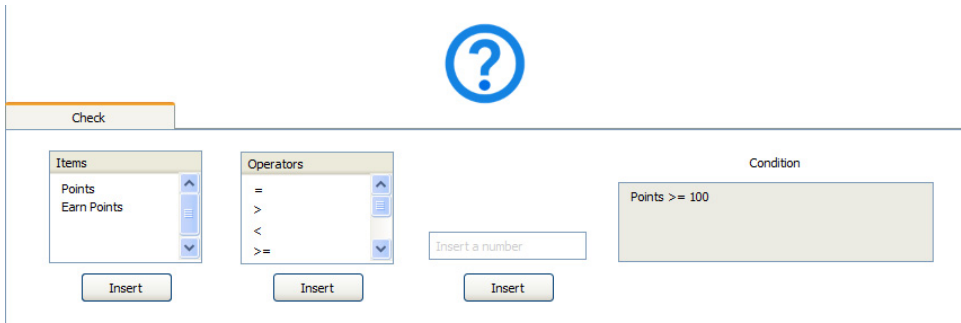


Fig. 5. The VULCAN UI for customizing the check operator.

the domain entities that can be used while managing the action logics, and includes automatically constructed permanent entities (i.e. those inferred by MODE during the generation of the data model) and those newly defined in the MODE mockup, within possibly different containers. Finally, the bottom area enables the end-users define properties of the operators being used within the workspace.

As an example, the specification of the control action shown in Fig. 7 is activated by clicking on the **update** action defined for the **Pay** button of the **Payment** page in Fig. 4. In particular, it specifies that the amount credit decreases by the *Total* amount spent, by means of the operation **Credit=Credit-Total**, whereas the update of the collected points is increased by the *Earn Points*, by means of the operation **Points=Points+Earn Points**. The currently selected symbol in the workspace window is colored in gold. As an example, in Fig. 7 the currently selected symbol is the one of the function for updating earned points, and its properties are shown in the bottom area.

However, more a complex logic could be associated to the control action, like introducing a check on the collected score, in order to reward the premium users upon exceeding a given score threshold, as shown in Fig. 8, where the user credit is increased by 10 at the cost of 100 points.

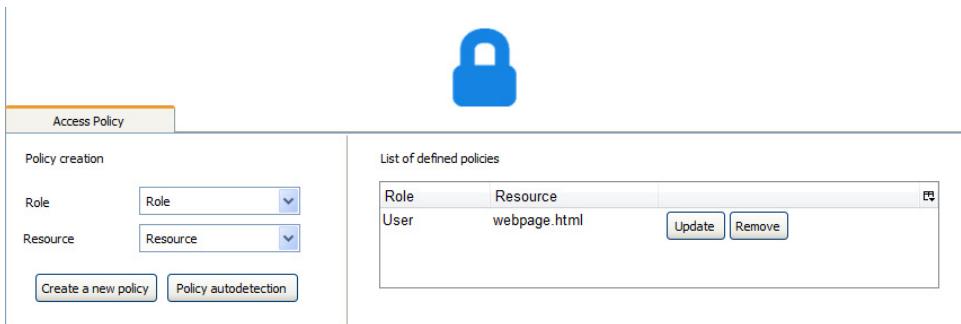


Fig. 6. The UI for customizing the access policy operator.

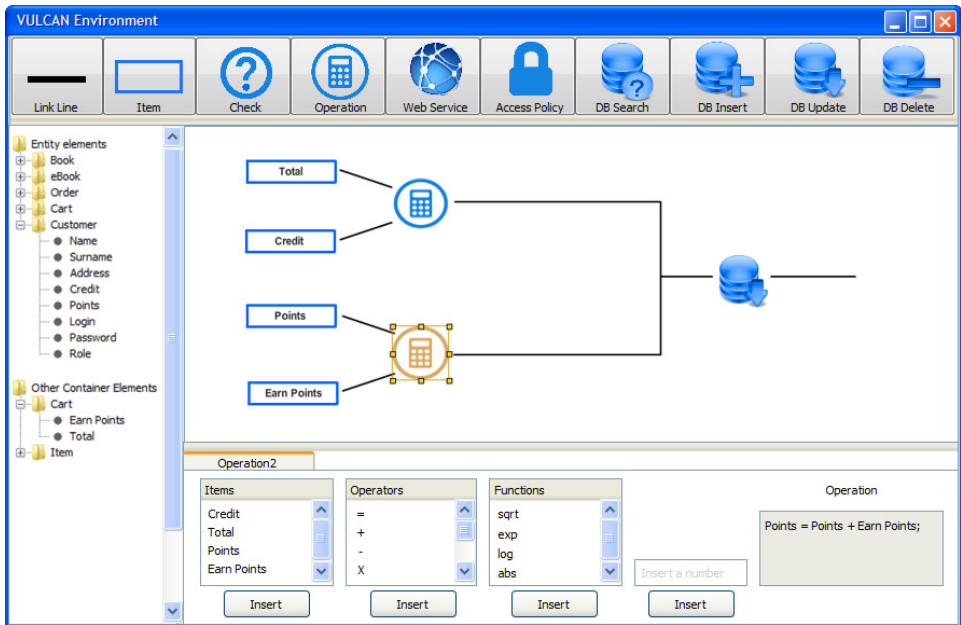


Fig. 7. The update action associated to the Pay button of Fig. 4.

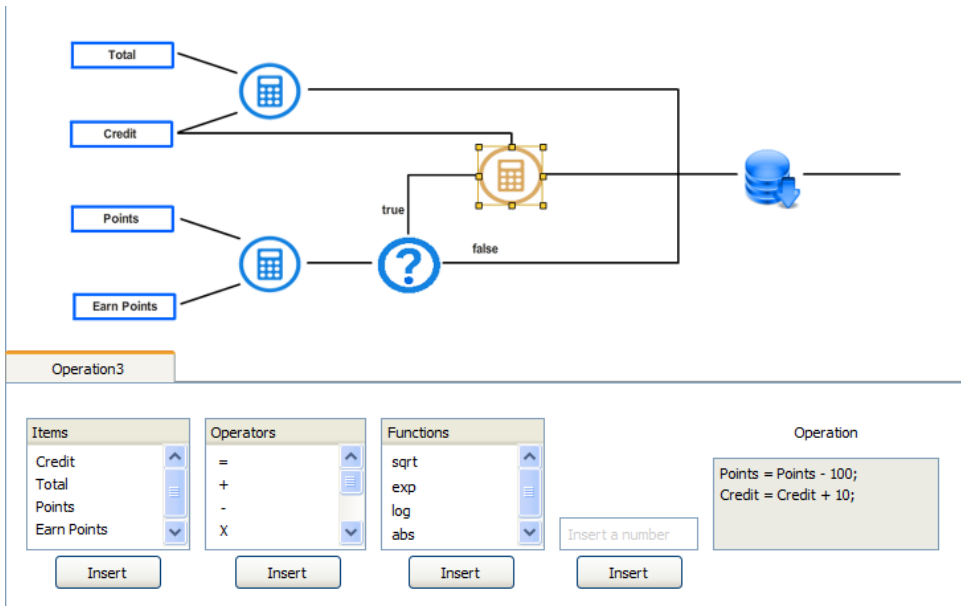


Fig. 8. An action associated to the Pay button of Fig. 4 that rewards premium users.

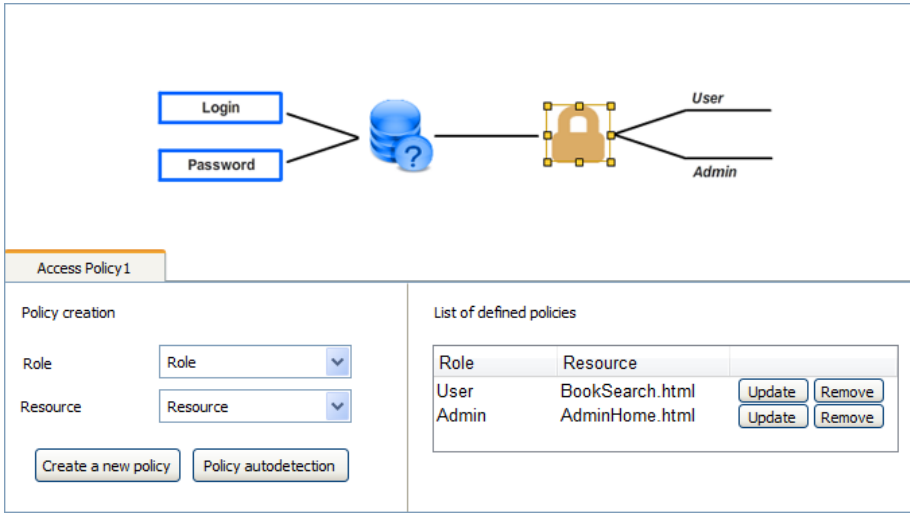


Fig. 9. The search action associated to the Sign in button in Fig. 4.

As said above, the VULCAN environment allows the user handle the control actions, also according to the restrictions defined for the different roles. As an example, in the Webapp presented in Fig. 4, the control action associated to the Sign in button in the Home Page could be described as shown in Fig. 9. There, the restrictions on the roles Admin

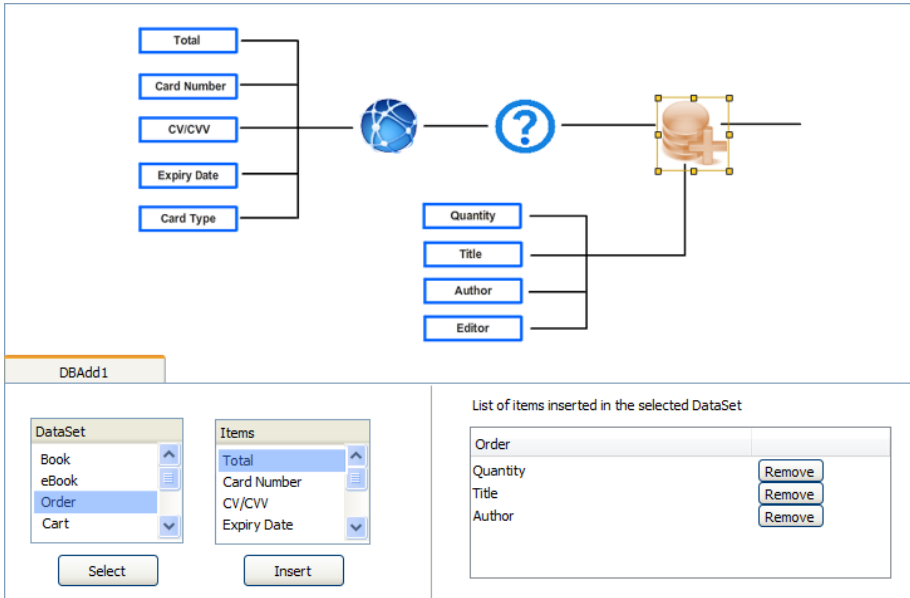


Fig. 10. The action associated to the Confirm button in Fig. 4.

and **User** cause a split on the associated control actions. In particular, after verifying their identities by means of their login and password, the role *User* will access the page `BookSearch.html`, whereas the role *Admin* will access the page `AdminHome.html`.

Figure 10 shows the control action triggered upon clicking the **Confirm** button in the page `Payment Details` (see Fig. 4). First, the action exploits a web service to check validity of the credit card info. Then, in case of validity, it completes the purchase operation, updating the database of sales.

It is important to notice that the end-user can avoid specifying the application logic, if not necessary. Moreover, it is not always necessary to use the **VULCAN** environment, since for simple Webapps it is sufficient using the **MODE** environment alone. In fact, in a typical EUD scenario the main goal is to gain high simplification, aiming to build small applications with little effort. However, when the user becomes more familiar with system characteristics, s/he would probably wish to face more complex problems, for which it will be necessary to provide details on the application logic behind each action (skill growth).

6. Generation of Web Applications

The system modules we use to implement the proposed EUDWeb development process are standalone ones, and currently communicate through file exchange. In particular, the **MODE** environment embeds **Balsamiq** to create mockups of the Webapps [1], and is capable of reading the **RODE** generated XML file containing the mappings between roles and colors. The modules **RODE** and **VULCAN** have been implemented in Java by using Aspect Oriented Programming (AOP), and in particular, the following technologies: Eclipse, JBoss application server, AspectJ, Java Authentication and Authorization Service (JAAS), and OMG Resource Access Decision (RAD) facility. They cooperate to produce the XACML policy files, and the **VULCAN** specification is exploited by **MODE** for updating the control flow of the generated controllers, according to the specified control actions.

The XML version of the mockups produced by **Balsamiq** is used to translate them into their equivalent tree data structure. The latter is important for two reasons: firstly, during the generation of the Webapp it helps performing frequent operations on the UI, such as searching a widget, identifying a group of widgets, checking uniqueness of a group of widgets, and finding source and destination of navigation widgets; secondly, during the evolution of the Webapp it checks against the Webapp tree structure the effects of changes to UI elements, prior committing them to the structure and behaviour of the Webapp. The tree data structure is also used to infer the data model of the Webapp, and to generate the whole Webapp by using client side MVC components, and server side MC components [12].

The MVC-MC architecture generated with the proposed approach is shown in Fig. 11. The MVC-MC pattern has been designed by following the principle of optimal separation of concerns. The main concerns are the separation of components within the client-side MVC, and within the server-side MC. Since a Webapp has both

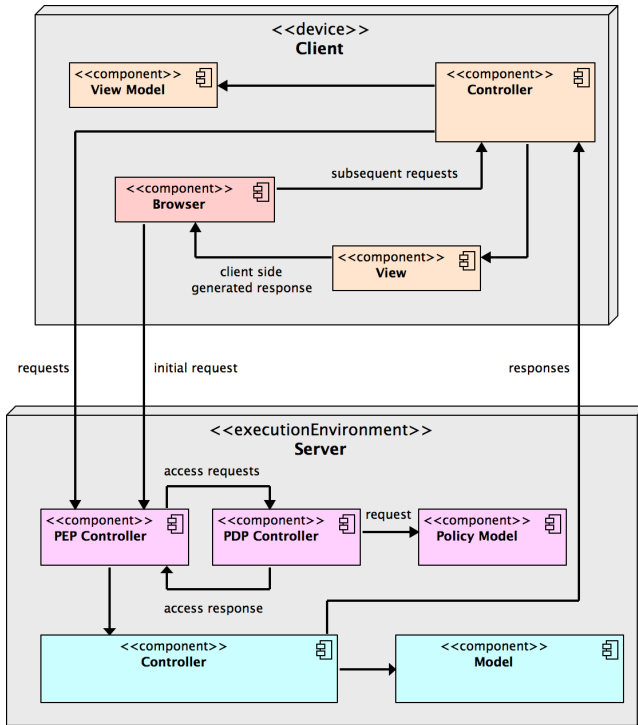


Fig. 11. The architecture of the generated Webapp.

client-side and server-side processing components, the MVC paradigm has been suited to the two sides. However, in our architecture the server side View component has been omitted to reduce the design complexity of the generated Webapp. That is, the generated Webapp will always create dynamic HTML by using the View logic on the client side. Secondly, beyond the initial request, all the communications between the two sides are managed through requests from client-side to server-side controllers. This eliminates the need of persistent connections between the server and the client. Similarly, beyond the initial request, all the communications between the user and the Webapp are handled by the client-side controller.

As shown in the Fig. 11, on the server side there are components controlling access to resources, i.e. *PEP* (Policy Enforcement Point), and *PDP* (Policy Decision Point) controllers, and a component managing policies, i.e. the *Policy Model*. In particular, all the access requests coming from the client are intercepted by the PEP, which issues a request to the PDP component. The latter evaluates such request against the policy stored within the mockup generated XACML file contained in the Policy Model, and returns a response to the PEP, which in turns grants or deny access to the requested resource. The message exchange underlying this process is detailed in Fig. 12.

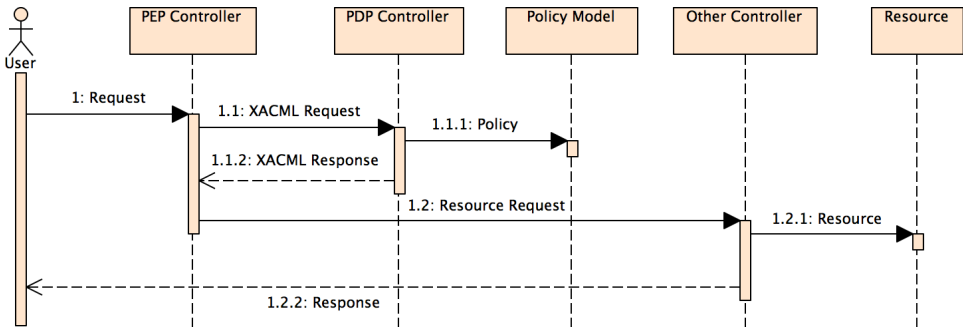


Fig. 12. The sequence diagram describing the messages exchanged between the Webapp components devoted to access control.

7. Usability Evaluation

The goal of the presented EUDWeb framework is to enable end-users to create web applications embedding access control mechanisms. In order to reach such goal we targeted three sub-goals:

- (1) the MODE environment has to address the limitations of current mockup tools, since they lack end-user paradigms for the specification of access control mechanisms in Webapp development;
- (2) the VULCAN environment has to enable end-users specify control actions by using metaphors, since the targeted users would not be able to use coding;
- (3) the whole EUDWeb framework has to enable end-users to accomplish the task of building a complete Webapp including access control mechanisms.

In this section, we report on the evaluation of the presented EUDWeb framework, involving end-users in a real usage of its different environments. In particular, since to the best of our knowledge there is no tool with analogous functionalities, as also stated in Sec. 2, we could only perform an evaluation based on the capability of the presented EUDWeb framework to reach the previous stated subgoals, and in particular subgoal (3). A comparative evaluation could only be accomplished by manually simulating the specification of access control mechanisms within existing WYSIWYG environments. However, this could not be effectively made in practice, since the proposed EUDWeb framework is targeted at users with little technological background. Moreover, such a manual comparison would be of limited value since it would not allow to measure important parameters, such as performances, easy of use, error-proneness.

The evaluation was performed by involving eight users with an average age of 38.37 years (standard deviation = 11.18), no programming skills, and with an even distribution of both genders. Table 2 reports their profiles. All participants underwent a three hours tutorial on access control concepts, mockup-driven web development, and the proposed EUDWeb framework. The evaluation was conducted by a

Table 2. Participant’s profiles.

| ID | Background | Age | Gender |
|-------|------------------------|-----|--------|
| User1 | Industrial Manager | 45 | F |
| User2 | Student of Law | 19 | F |
| User3 | Math Teacher | 48 | M |
| User4 | Architect | 41 | M |
| User5 | Student of Archaeology | 25 | F |
| User6 | Web Master | 35 | M |
| User7 | Lawyer | 44 | M |
| User8 | Municipality | 50 | F |

research fellow using two data collection methods: observation and questionnaires. The duration of each test session ranged from one up to four hours, and the goals of the evaluation were clearly stated to participants.

7.1. Method

The user evaluation included an introductory questionnaire, a set of tasks to accomplish, and a post-test questionnaire.

The introductory questionnaire included: demographic information, computer skills, previous EUD experiences, and access control backgrounds. This step was performed to categorize end-users and collect data on their backgrounds.

After undergoing the introductory questionnaire, participants accomplished a usability test consisting of four test sessions, in each of which they were requested to execute a different task by using the proposed EUDWeb framework. Tasks included:

- (a) Creation of a simple Webapp and evaluation of VULCAN operators;
- (b) Creation of a Webapp exploiting a web service to show the weather forecasts based on an input city;
- (c) Creation of a Webapp enabling generic users to register their credentials to a website, and successively access a welcome page. The administrator can access the system for viewing the registered users;
- (d) Creation of a Webapp for selling products. It allows administrators to add products, companies to buy and add products, and customers to buy products.

Finally, participants underwent a post-test questionnaire, which included a quantitative and a qualitative analysis. The quantitative analysis included 23 sentences, and the user was asked to evaluate each sentence within a Likert scale (strongly disagree = 1, disagree = 2, neutral = 3, agree = 4, strongly agree = 5). The purpose of this set of sentences was to support the analysis of the following aspects: user friendliness of the environments, learnability, efficiency, memorability, effectiveness, and intuitiveness. The qualitative analysis of the post-test questionnaire focused on: best and worse characteristics on the EUDWeb framework, usage obstacles, improvement suggestions, general comments, and task accomplishment.

7.2. Results

The results showed that the framework could be used to complete the assigned tasks with a moderate effort. The framework enabled the user to proficiently accomplish most relevant and critical tasks, especially the specification of control actions triggered by UI events, and in particular, those related to access control. This corroborates the effectiveness of the metaphor underlying the visual language of VULCAN.

In what follows, after presenting the results of the introductory questionnaire, we discuss performances of participants in the accomplishment of the assigned tasks, and finally present the results of the post-test questionnaire.

From the analysis of answers to the introductory questionnaire, it came out that all users had familiarity with computers, internet, and main office automation packages. However, concerning web development, two participants had low level skills, two medium level, and the remaining had no skills. Four participants were familiar with the concept of EUD, and each of them had previously used some software package supporting such paradigm, such as formulas or macros in Microsoft Excel, Microsoft FrontPage, and Dreamweaver. Among these, two also had been exposed to access control problems, together two additional participants.

Regarding the performances of participants in the accomplishment of tasks, we evaluated them according to the following scale: 0, not completed; 1, completed with difficulty or help; 2, easily completed. The first task was completed by all the participants, even though User7 and User8 had some difficulties. Similar results were achieved for the second task, but also User5 required some assistance. Tasks 3 and 4 involved the specification of control actions concerning access control. All participants completed task 3, but half of them requested some assistance. Finally, one participant was not able to complete task 4, and three of them requested some help. We observed that User7 and User8 needed assistance for all the tasks, and more specifically, on the usage of tool features. Conversely, the participants with some knowledge on EUD and web development (User1, User3, and User6) were proficient in all the assigned tasks, but User4, who needed some assistance on task 4. Participants with less pertinent skills (User2 and User5) achieved different performances. In particular, User5 requested help only for tasks 2 and 3, whereas User2 did not complete task 4, and requested assistance for task 3.

Finally, we performed a quantitative and qualitative analysis by means of the post-task questionnaire. The quantitative analysis aimed at evaluating the user friendliness of the environments, their learnability, efficiency, memorability, effectiveness, and satisfaction while using the EUDWeb framework. Box plots built from the participants answers on a Likert scale 1 to 5 are shown in Figs. 13 and 14. The box plot includes the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum) for each participant and each question, respectively. Bottom and upper tickers represent minimum and maximum values, respectively. Bottom and upper lines for the purple boxes represent Q1 and Q2, respectively. The upper line of the green box is the Q1.

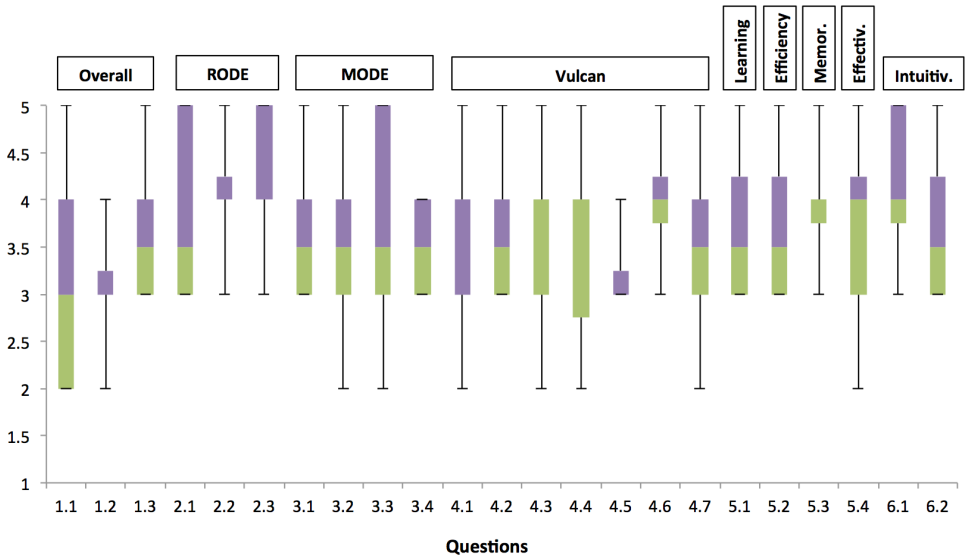


Fig. 13. Quantitative analysis results by questions.

The qualitative analysis highlighted some usage flaws related to the prototypal implementation, and to the poor integration among the environments of the framework. On the other hand, as a positive remark, most participants considered the metaphors underlying the environments intuitive and pleasant, making them sufficiently easy to use.

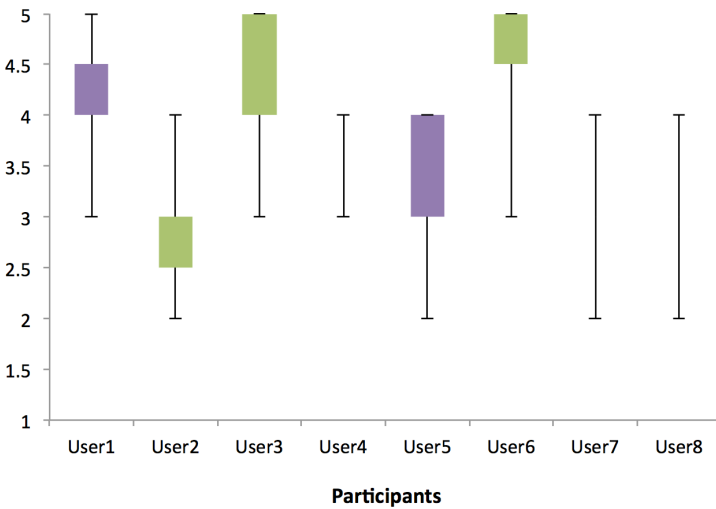


Fig. 14. Quantitative analysis results by participants.

7.3. Discussion

The usability evaluation discussed above confirmed that the metaphors underlying the environments of the proposed framework were intuitive and enabled the majority of end-users to accomplish Webapp development tasks, including those embedding some access control functionalities. The evaluation revealed some little problems on UI and on interoperability among the proposed environments, probably due to the prototypal implementation. Thus, we expect the tool became much more effective as the quality of the implementation grows. We could also observe that participants with some previous EUD knowledge could be more proficient. Moreover, participants showed no particular problems in the implementation of access control functionalities.

The quantitative analysis in Fig. 13 indicates that RODE was perceived as easier to use, with a mean of 4.08 and standard deviation of 0.77, with respect to MODE and VULCAN that gained a mean of 3.59 and 3.61, and standard deviation of 0.84 and 0.87, respectively. Our challenge here was to prove that the environments MODE and VULCAN could help end-users master the complexity of core activities. To this end, the answers to questions from 3.1 to 4.7 revealed that we satisfactorily reached this goal. Moreover, answers to last six questions revealed that the environments were sufficiently easy to learn, efficient, intuitive, mnemonic, and required the right time and the proper number of steps to accomplish the assigned tasks.

As said above, the results reported in Fig. 14, together with average values computed for each user, revealed that the whole framework could be more effectively used by people with some EUD knowledge, and/or with some exposure to similar problems in their professional environment. However, we think that these results can be considered satisfactory, since we gained an global average score of 3.68, with a standard deviation of 0.86. Moreover, while the active involvement of less experienced users typically yields a reduced expectation on the levels of sophistication of the Webapp, our end-users could accomplish sufficiently complex tasks.

Although the participants pointed out the necessity to improve the implementation quality of the UI, they provided positive feedbacks on the adequacy of the used metaphors, and on the capability of the whole framework to guide the user towards the completion of meaningful Webapp programming tasks. Finally, the participants also pointed out the necessity of better levels of integration among the different environments of the framework, but this did prevent them from proficiently accomplish most of the assigned tasks. The feature they liked most was the use of colors to immediately recall the minimum role requested to execute tasks, whereas the configuration of icons for database operation in VULCAN was the feature that most participants liked less.

8. Conclusions and Future Work

We have presented an EUDWeb framework and tool enabling end-users develop Webapps embedding RBAC-based access control policies. In particular, we have

extended a previous mockup-based approach and tool [10] by empowering it with visual languages for specifying RBAC policies.

In this work we have focused on access control, since it is a Webapp programming functionality that the end-user perceives as one of the most complex. However, since there are few additional critical aspects to make EUDWeb effective, in the future we would like to develop new metaphors and visual languages to broaden the range of complex tasks that can be tackled within our EUDWeb environment. Particular attention deserve the management of stateless protocols, database implementation, and more general aspects of Webapp control flow management. Finally, we aim to simplify the specification process, by reducing the manual activities and by introducing sketch-based interaction paradigms [8, 11].

References

1. Balsamiq, Balsamiq mockups, <http://balsamiq.com/>. last accessed: January 22, 2015.
2. D. Basin, J. Doser and T. Lodderstedt, Model-driven security: From UML models to access control infrastructures, *ACM Trans. Softw. Eng. Methodol.* **15**(1) (2006) 39–91.
3. I. Bouchrika, L. Ait-Oubelli, A. Rabir and N. Harrathi, Mockup-based navigational diagram for the development of interactive web applications, in *Proc. Int. Conf. Information Systems and Design of Communication*, 2013, pp. 27–32.
4. D. Bricklin, B. Frankston and D. Fylstra, VisiCalc, software arts. <http://www.bricklin.com/history/intro.htm>. 1979, last accessed: January 22, 2015.
5. L. Caruccio, V. Deufemia, C. D’Souza, A. Ginige and G. Polese, Supporting access control within a mockup-based EUDWeb environment, in *Proc. 7th International Symposium on Visual Information Communication and Interaction*, 2014, pp. 88–97.
6. S. Chang, G. Polese, R. K. Thomas and S. Das, A visual language for authorization modeling, in *Proc. IEEE Symposium on Visual Languages*, 1997, pp. 110–118.
7. O. Chudnovskyy, T. Nestler, M. Gaedke, F. Daniel, J. I. Fernández-Villamor, V. Chepegin, J. A. Fornas, S. Wilson, C. Kögler and H. Chang, End-user-oriented telco mashups: The OMELETTE approach, in *Proc. 21st International Conference on World Wide Web*, 2012, pp. 235–238.
8. G. Costagliola, V. Deufemia, G. Polese and M. Risi, A parsing technique for sketch recognition systems, in *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing*, 2004, pp. 19–26.
9. A. Coyette and J. Vanderdonckt, A sketching tool for designing anyuser, anyplatform, anywhere user interfaces, in *Human-Computer Interaction — INTERACT 2005*, Lecture Notes in Computer Science, Vol. 3585, 2005, pp. 550–564.
10. V. Deufemia, C. D’Souza and A. Ginige, Visually modelling data intensive web applications to assist end-user development, in *Proc. 6th International Symposium on Visual Information Communication and Interaction*, 2013, pp. 17–26.
11. V. Deufemia, M. Risi and G. Tortora, Sketched symbol recognition using latent-dynamic conditional random fields and distance-based clustering, *Pattern Recognition* **47**(3) (2014) 1159–1171.
12. C. D’Souza and A. Ginige, MVC-MC: A rich internet application architecture for optimal separation of concerns, in *Int. Conf. Computer and Software Modeling*, 2010, pp. 78–82.
13. D. F. Ferraiolo, R. D. Kuhn and R. Chandramouli, *Role-Based Access Control* (Artech House, Norwood, 2007).

14. M. Giordano and G. Polese, Visual computer-managed security: A framework for developing access control in enterprise applications, *IEEE Software* **30**(5) (2013) 62–69.
15. M. Giordano, G. Polese, G. Scanniello and G. Tortora, A system for visual role-based policy modelling, *Journal of Visual Languages and Computing* **21**(1) (2010) 41–64.
16. A. Heydon, M. Maimone, J. D. Tygar, J. Wing and A. Zaremski, Miró: Visual specification of security, *IEEE Trans. Software Engineering* **16**(10) (1990) 1185–119.
17. J. A. Hoagland, R. Pandey and K. N. Levitt, Security policy specification using a graphical approach, Technical report, University of California, 1998.
18. A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw and S. Wiedenbeck, The state of the art in end-user software engineering, *ACM Comput. Surv.* **43**(3) (2011) 21:1–21:44.
19. M. Koch, L. V. Mancini and F. Parisi-Presicce, A graph-based formalism for RBAC, *ACM Trans. Inf. Syst. Secur.* **5**(3) (2002) 332–365.
20. B. W. Lampson, Protection, *SIGOPS Oper. Syst. Rev.* **8**(1) (1974) 18–24.
21. J. Lin, J. Wong, J. Nichols, A. Cypher and T. A. Lau, End-user programming of mashups with Vegemite, in *Proc. 14th International Conference on Intelligent User Interfaces*, 2009, pp. 97–106.
22. D. Lizcano, F. Alonso, J. Soriano and G. López, A new end-user composition model to empower knowledge workers to develop rich internet applications, *J. Web Eng.* **10**(3) (2011) 197–233.
23. B. A. Myers, *Creating User Interfaces by Demonstration* (Academic Press, San Diego, 1998).
24. B. A. Myers and W. Buxton, Creating highly-interactive and graphical user interfaces by demonstration, *SIGGRAPH Comput. Graph.* **20**(4) (1986) 249–258.
25. T. Nestler, A. Namoun and A. Schill, End-user development of service-based interactive web applications at the presentation layer, in *Proc. 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 2011, pp. 197–206.
26. J. Nichols and T. Lau, Mobilization by demonstration: Using traces to re-author existing web sites, in *Proc. 13th International Conference on Intelligent User Interfaces*, 2008, pp. 149–158.
27. OASIS, Oasis extensible access control markup language (XACML) v2.0 specification, <http://www.oasis-open.org/committees/xacml/>, last accessed: January 22, 2015.
28. T. O'Reilly, What is web 2.0? Design patterns and business models for the next generation of software, <http://oreilly.com/web2/archive/what-is-web-20.html>, 2005, last accessed: January 22, 2015.
29. F. Pérez, P. Valderas and J. Fons, Towards the involvement of end-users within model-driven development, in *Proc. 3rd International Conference on End-user Development*, 2011, pp. 258–263.
30. J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna and N. Koch, Improving agility in model-driven web engineering, in *CAiSE Forum*, Vol. 734 of *CEUR Workshop Proceedings*, 2011, pp. 163–170.
31. J. M. Rivero, G. Rossi, J. Grigera, J. Burella, E. R. Luna and S. Gordillo, From mockups to user interface models: An extensible model-driven approach, in *Proc. 10th International Conference on Current Trends in Web Engineering*, 2010, pp. 13–24.
32. J. Rode, M. B. Rosson and M. A. Pérez Quiñones, End user development of web applications, in *End User Development*, Vol. 9 of *Human-Computer Interaction Series*, 2006, pp. 161–182.

33. R. Sandhu, D. Ferraiolo and R. Kuhn, The NIST model for role-based access control: Towards a unified standard, in *Proc. 5th ACM Workshop on Role-Based Access Control*, 2000, pp. 47–63.
34. R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. E. Youman, Role-based access control models, *Computer* **29**(2) (1996) 38–47.
35. H. Störrle, Model-driven development of user interface prototypes: An integrated approach, in *Proc. 4th European Conference on Software Architecture*, 2010, pp. 261–268.
36. R. K. Thomas and R. S. Sandhu, Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management, in *Proc. IFIP TC11 WG11.3 11th International Conference on Database Security XI: Status and Prospects*, 1998, pp. 166–181.
37. M. Toomim, S. M. Drucker, M. Dontcheva, A. Rahimi, B. Thomson and J. A. Landay, Attaching UI enhancements to websites with end users, in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1859–1868.
38. F. Valverde and O. Pastor, Facing the technological challenges of web 2.0: A RIA model-driven engineering approach, in *Proc. 10th International Conference on Web Information Systems Engineering*, 2009, pp. 131–144.
39. N. Zhang, M. Ryan and D. P. Guelev, Synthesising verified access control systems in XACML, in *Proc. 2004 ACM Workshop on Formal Methods in Security Engineering*, 2004, pp. 56–65.

Copyright of International Journal of Software Engineering & Knowledge Engineering is the property of World Scientific Publishing Company and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.