Реализация на динамична структура на приложение в интерактивна среда

Implementation of a Dynamic Application Structure in a Real Time Framework

Samuil Nikolov, Anatoliy Antonov

Резюме: Статията представя проучванията на авторите върху теортичния модел и разработването на система върху която е разработена интерактивна среда за динамично управление на приложенията. Разгледан е граф, представящ структурата на скриптово-базирано приложение. Представени са правила за промяна на структурата и начини за динамично генериране. Показани са принципите за създаване на вътрешната структура на базата на шаблони. Демонстриран е пример за динамично създаване на система за контрол на достъпа до приложението.

Ключови думи: Управление на динамични приложения, Среда за създаване на приложения, Създаване на динамична структура

Abstract: The paper presents the authors' research on the theoretical model and experience in implementing a real system, on which a framework for dynamic application control is based. A graph is considered, representing the structure of scripts- based application. Rules for modifying its structure and ways of dynamic generation are discussed. Principles for building the internal structure of the subtasks are shown, based on object identifiers and template instances. An example application is demonstrated showing how to create dynamically user access control.

Keywords: Dynamic application control, Application framework, Creation of dynamic system structure

I. INTRODUCTION

Specific software applications have one major disadvantage – their functionality is highly limited by the area for which they have been developed. The complexity of computer and communication systems makes it difficult for even the original system developers to analyze, model, or predict system behavior, let alone anticipate the emergent behavior of multiple interacting systems [9]. Often an entirely new project is needed for the development of software with similar tasks in real life and most of the working code has to be rewritten just because of minor specifics that come to be crucial for the entire work of the program and its usefulness in other cases. Building executable models need theory, methods, and tools for modeling complex heterogeneous systems [1]. During the development of software technologies, several ways of reusing program code were developed [3]. One of them is the development of libraries with compiled code, containing a set of functions with general purpose, that could be linked either directly to the code of the developed Application (statically linked libraries) or used and distributed from the program as separate files (dynamically linked libraries) [4], [5]. Other way of reuse is supporting and distribution of software code, in most cases as open source in internet [8]. The disadvantage in this way of reuse is that the code is language-specific. A third option of faster and effective development is the development of general-purpose programs – frameworks - that are used as a base for specific applications. Usually they are created by supplying small script-language files [2], [6] that define the specific purpose of the final application. This publication presents the principles of one such commercial framework. Every application developed with the framework is constructed from subsystems (modules) that represent subtasks from the main problem to be solved. The subsystems contain dynamic lists of object identifiers (items) and have a dynamic set of associated templates (script models) that can be applied to every one of them. Every template defines user interface and business logic that helps to serve one of the purposes of the application [5], [7]. The connection between the subtasks – the plan of their execution is dynamically carried out by the framework according to the script, predefined in the templates. The execution path is a result of certain end-user actions on the dynamically generated user interface.

II. DYNAMIC APPLICATION STRUCTURE

Every user-defined application over the framework can be examined as a graph G, which nodes are the structural subsystems (modules). They are linked dynamically by control transitions that supply a list of object identifiers and a list of models that can be applied to them.

$G = \{ N , E \}$, where (1)

$N = \{ S_1, S_2, .. , S_n \}$ – set of subsystems S_i

$E = \{ <S_i , S_j, ItemList_k > | i \neq j\}$ – set of connections

By allowing changes to the subsystems and their connections, the framework provides means for creating flexible and easy to modify applications.

$| E | \neq$ const (2)

$| N | \neq$ const

Figure 1 represents a system, comprised from three subsystems – S_1, S_2 ,S_3, with their associated lists of models and lists of items.

$N=\{ System, S_1, S_2, S_3\}$ (3)

$E=\{ <System, S_1$, [Item1.1, Item1.2, Item1.3]>,

<System, S_2, [Item2.1,Item2.2]>,

<System, S_3,[Item3.1,Item3.2,Item3.3]>,

<S_3, S_2, [Item2.1,Item2.2]>,

<S_2, S_1, [Item1.1,Item1.2,Item1.3]> }

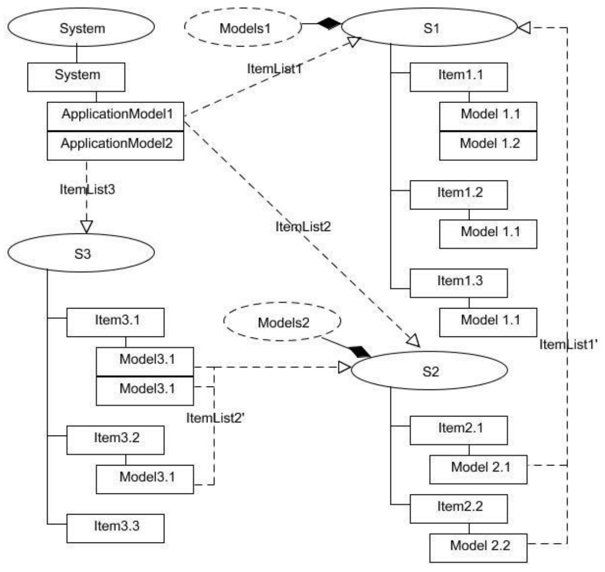


Figure 1. Example structure of a system

The initial node of the graph is the reserved subsystem System, containing one reserved identifier - System – the starting point for application execution. It is used to instantiate the models that will control the further operation of the subsystems and their interaction with one another.

1. Definition of a subsystem

Every subsystem contains a list of object identifiers that are connected to the actual problem being solved by the subsystem. It has a variable set of templates that are associated with it. The set of templates can be extended during application development.

$S = \{ E_i , M_j | \ i= 1..n, j= 1..m\}$

E_i – object identifier (a string) (4)

M_j – model template

The end user can add, remove and modify the list of identifiers. Modifying it means instantiating a new model from the supplied list for a certain identifier and filling the model's data with new values.

2. Dynamic changes to the application structure

New subsystems can be created dynamically by adding items to the subsystems set and connecting them by changing the set of edges.

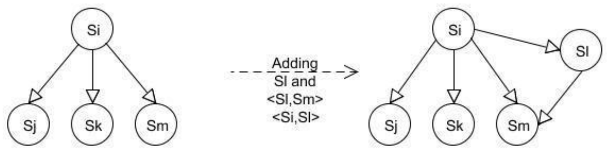


Figure 2. Adding a subsystem Sl to Si and Sm

- The process of adding subsystems is controlled by script variables, activated by the end user. Usually these are represented by simple UI checkboxes;
- Associating new connections between the subsystems consists of predefining the variables controlling the subsystem transitions;
- Deleting a subsystem is done by deleting it from the set and removing all the connections leading to and from it

3. Template (model script)

The template defines a context for every object identifier, supplied to the subsystem and contains the following elements:

- definition of the user interface of a concrete subsystem task;
- a set of variables, associated to the user interface items that the end user can input into:

$$V = \{ V_i \mid i = 1..n \} \; ; \qquad (5)$$

- a set of rules F for changing the values of the variables:

$$P = \{ P_i \mid i = 1..k \} \qquad (6)$$

Every rule has the following form:

$$V_i \leftarrow F (V_1, V_2 \ldots V_n), \qquad (7)$$

where V_i belongs to the set of variables;

- Rules for modification of user interface. They allow changing the appearance of user controls on the screen including hiding, showing, changing read only status etc. [6] The means for designing and generating the user interface, associated to the template is not subject of the current publication;
- Rules for transition to another subsystem ψ. They allow designing the transition from the current subsystem to another one(Si) or to the same subsystem under different conditions – like another list of associated items:

$$\psi (S_i, [Item1,Item2,...,Itemn])$$

$$S_i \in \{ S_1, S_2, .., S_n \} \qquad (8)$$

The end user is able to create new instances of templates for every identifier and input data in them concerning either the specific task of the subsystem, or preparations for transition to another subsystem.

4. Creating the structure of an example application

To illustrate the principles of creating and dynamically controlling subsystems, we will present the application defining the user access to the framework. All applications start from a common root subsystem – System, containing single item – RFW System. The application itself consists of two subsystems – registering user roles and registering users:

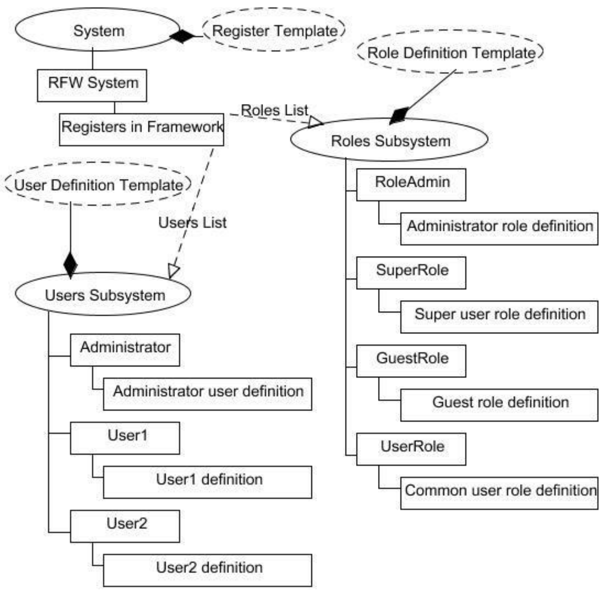
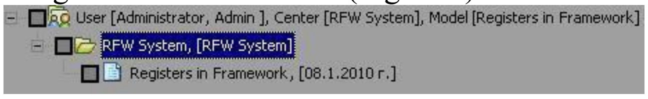


Figure 3. Structure of the example application for defining access control

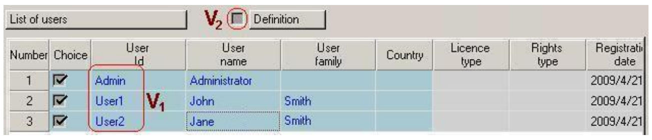
To define the new user access control application, a specifically designed template – Register template, is supplied that defines the transitions and an entry point for the application. An instance of this template is created for the item RFW System, called "Registers in Framework" (Figure 4).



User [Administrator, Admin], Center [RFW System], Model [Registers in Framework]
RFW System, [RFW System]
Registers in Framework, [08.1.2010 г.]

Figure 4. Initial subsystem

The Register template defines two user interface tables – a list of users and a list of user roles. On Figures 5 and 6 are shown table contents after adding the instance and opening it for input:

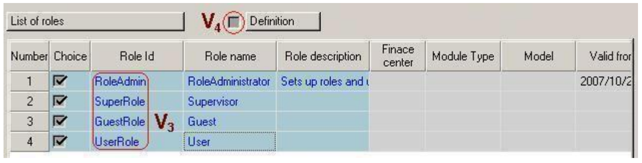


List of users V_2 Definition

Number	Choice	User Id	User name	User family	Country	Licence type	Rights type	Registrati date
1	☑	Admin	Administrator					2009/4/21
2	☑	User1	John	Smith				2009/4/21
3	☑	User2	Jane	Smith				2009/4/21

V_1

Figure 5. Table containing a list of users



List of roles V_4 Definition

Number	Choice	Role Id	Role name	Role description	Finace center	Module Type	Model	Valid fror
1	☑	RoleAdmin	RoleAdministrator	Sets up roles and u				2007/10/2
2	☑	SuperRole	Supervisor					
3	☑	GuestRole	Guest					
4	☑	UserRole	User					

V_3

Figure 6. Table containing list of user roles

Both tables are user-extendable, i.e. the end user can add new lines to them and input the ids, names and other optional descriptions for both users and roles. This initial template model is used by the system administrator to specify the users after installation.

All template instances are persisted in a data base. The tables used are designed in such a manner as to reflect the flexibility of the framework system, but their description is not subject to the current publication.

From application-structure point of view there are four essential variables V_1-V_4:

- V_1 – a list of user identifiers, linked to the table's column
- V_2 – active interface control, used to trigger the connection between the main subsystem System and the Users Subsystem.
- V_3 – a list of role identifiers, linked to the table's column
- V_4 - active interface control, used to trigger the connection between the main subsystem System and the Roles Subsystem.

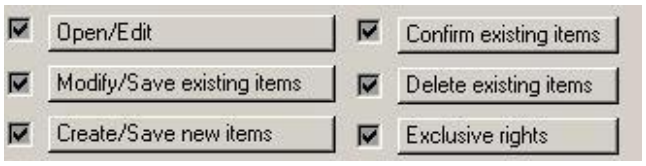
Besides the user interface definition, the template also contains definitions of transition rules:

$$\Psi \text{ (User Subsystem, } [V_1]) \qquad (9)$$

$$\Psi \text{ (Role Subsystem, } [V_3])$$

If V_2 is activated, the framework will transfer the control to the User Subsystem using the list of user identifiers V_1. The same approach is used when activating the Role Subsystem transition rule.

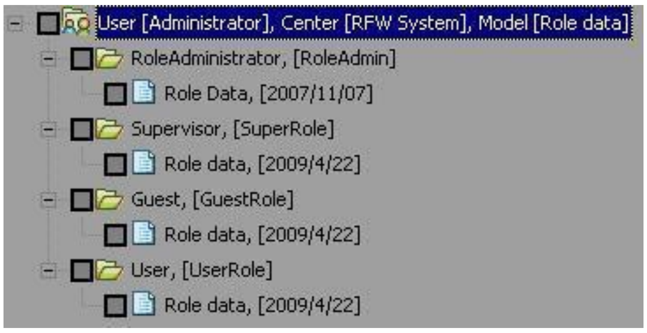
Each of the two subsystems has an associated template. The role definition template (see figure 3) contains a set of interface elements (in this case checkboxes), allowing the selection of different actions that are allowed for the specifically defined role. Those can be any set that could be interpreted during the usage of the security system, for example (Figure 7):



☑ Open/Edit
☑ Confirm existing items
☑ Modify/Save existing items
☑ Delete existing items
☑ Create/Save new items
☑ Exclusive rights

Figure 7. Possible access rights that can be defined in the framework

It is fully extendable, i.e. any access type that is interpretable by the framework can be included just by adding a check box to the template model script. The template also contains other role definition data like tables for specifying a subset of modules for which the access rights are valid, if such restrictions are necessary. When the Role Subsystem transition rule (9) is fired, on user activation of the V_4 element (Figure 6), the initial subsystem System transfers control to the Role Subsystem and the role identifiers are supplied as framework items. For every item, it is possible to create a definition – template instance that will specify what the role can or cannot perform in the system (Figure 8):

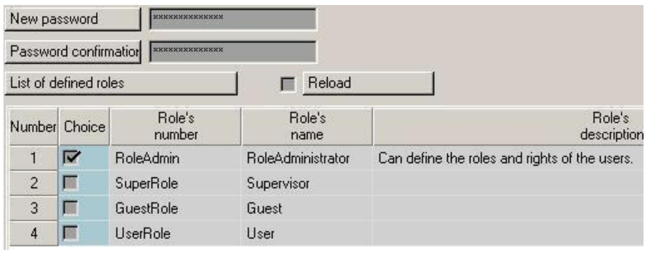


```
User [Administrator], Center [RFW System], Model [Role data]
    RoleAdministrator, [RoleAdmin]
        Role Data, [2007/11/07]
    Supervisor, [SuperRole]
        Role data, [2009/4/22]
    Guest, [GuestRole]
        Role data, [2009/4/22]
    User, [UserRole]
        Role data, [2009/4/22]
```

Figure 8. List of identifiers for the Role Subsystem with specific instances of the Role definition template

Each template instance is persisted with different combination of checked checkboxes to reflect the specific role access rights.

The other template, for the User Subsystem consists of a user interface table (Figure 9) with role identifiers and a field for entering the user's password:



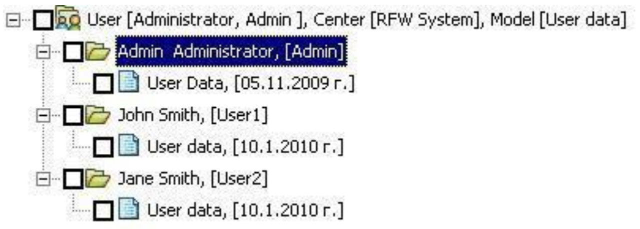
New password	***************
Password confirmation	***************
List of defined roles	☐ Reload

Number	Choice	Role's number	Role's name	Role's description
1	☑	RoleAdmin	RoleAdministrator	Can define the roles and rights of the users.
2	☐	SuperRole	Supervisor	
3	☐	GuestRole	Guest	
4	☐	UserRole	User	

Figure 9. UI of the template for user definition

The table is filled from the persistent source with the role data model instances that have been defined in the Role Subsystem. The table has a column of checkboxes that allow the administrator to select the desired roles for the user being defined.

Similarly, to the Role Subsystem transition, when the user activates the V_2 element, the initial subsystem System transfers control to the User Subsystem and the user identifiers that were inserted in the users table (Figure 5) are set as items and can be defined by instantiating user definition templates for each one of them:



```
User [Administrator, Admin ], Center [RFW System], Model [User data]
    Admin Administrator, [Admin]
        User Data, [05.11.2009 г.]
    John Smith, [User1]
        User data, [10.1.2010 г.]
    Jane Smith, [User2]
        User data, [10.1.2010 г.]
```

Figure 10. List of identifiers for the User Subsystem with specific instances of the User Definition template

The template instances are filled and persisted for every user with his own, unique, combination of roles. Thus, every user that logs on the system has a defined set of actions allowed or denied.

III. CONCLUSIONS AND FUTURE WORK

The principles of creating the described framework show the possibilities of more direct and quick design and realization of projects and their execution in a universal medium. The building blocks of the project, the subtasks that are defined, can be linked dynamically thus solving the common task. Every subtask can associate a list of object identifiers and a list of models that can be used to define the solution of the specific problem. The described approach was used by the authors to build various applications. A more complex example of such is a financial portfolio evaluation application that consists of subsystems for defining stock indexes, instruments, positions, portfolio filters, portfolio lists, the portfolios themselves as well as a subsystem for applying various analyses on the defined portfolios. They all work together by using each other's data stored in the database, where the defined instruments use the data about the stock indexes, the defined portfolios use data from filters and lists to select various dynamic instrument subsets as their members and the analysis modules use the data stored for portfolios to evaluate and optimize the expected profits.

ЛИТЕРАТУРА:

[1] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis Model Checking: Algorithmic Verification and Debugging, Communications of the ACM, november 2009, Vol. 52(11): 75-84

[2] Giarratano, Joseph C., Ph.D. CLIPS User's guide Version 6.20, March 31st, 2002

[3] McConnel, Steve, Rapid Development: Taming Wild Software Schedules,1996

[4] Meyer, Bertrand, Reusable Software: The Base Object-Oriented Component Libraries, Prentice Hall, 1994

[5] Microsoft MSDN

[6] Paskalev Pl., Nikolov Vl. , Multi-platform, script-based user interface, CompSysTech, 2004

[7] Richter, Jeffrey, Applied Microsoft .NET Framework Programming, Microsoft Press, 2002

[8] http://en.wikipedia.org/wiki/Open_source_software

[9] JEANNETTE M. WING FIVE DEEP QUESTIONS IN COMPUTING, Communications of the ACM, january 2008, Vol. 51(1): 58-61

За контакти:
Инж. Самуил Николов
д-р Анатолий Антонов
Еврориск Системи ООД
9002 Варна, ул. "Генерал Киселов "31
e-mail: antonov at euroriskSystems dot com

Рецензент:
доц. д-р. инж. Елена Рачева- ТУ-Варна