A COMPONENT SPECIFICATION LANGUAGE FOR THE ENTERPRISE SOFTWARE

Adrian Alexandrescu
"Ovidius" University/ Mathematics and Informatics Department, bd. Mamaia Nr.124, Constanta
aalexandrescu@univ-ovidius.ro

Abstract: *The paper presents a Component Specification Language (CSL) used in the development process of an enterprise information system or software application (referred as enterprise software). The components correspond to the main concepts of the enterprise domain, which will be represented within the enterprise software. The enterprise software will be considered, event processing software. The concepts of the enterprise domain have been grouped in four categories: events, actions, entities and states. The concepts, and the related components, have some characteristics called attributes. the attributes have at their turn properties. The Component Specification Language enables the description of the software components, this meaning: the category specification (entity, event, action and state), the attributes specification and the properties specification for the attributes, resulting a CSL software description. The CSL software description may be automatic translated into a relational database schema and a set of classes for the component user-interfaces. The paper also describes the automatic translation process.*

Keywords: enterprise information system, enterprise software application, conceptual modeling, component description language, automatic translation

1. Introduction

The enterprise software development is a process that consumes a lot of resources. Usually, the development process contains the software requirements specification, the domain concepts identification, the design of some models for these concepts and the relationships between them, the implementation of the models using programming languages, and the maintenance of the resulting products. This article proposes the use of a Component Specification Language (CSL) for the description of the enterprise software components, and an automatic translation of these descriptions into final software representations. This approach enables the developer to concentrates on defining domain concepts, rather than finding solutions for designing and implementing models which may be done better by a translator. In this way we save a lot of time for software development and also enhance the quality of the generated components (theoretically, there are error free, and do not require testing). The maintenance of the software is easier because the modification of a concept need only to modify the description of the corresponding component in CSL and running the automatic translator. Adding a new concept, means only to do the description of its corresponding component, in CSL and requiring the automatic translation to obtain the software component.

2. The Component Specification Language (CSL)

CSL is a language which enables the description of the components of the enterprise software, corresponding to concepts from the enterprise software domain. CSL identifies four categories of concepts: events, actions, entities and states. The events appear, in the enterprise activity at different moments, and they are in general, descriptions of the usage and/or transformation of the enterprise resources, at a given moment. Optional, there are associated to an event one or more sequences of actions.

The events and the related actions, refer concepts from the enterprise domain, which we'll name entities. The entities can be things, beings or abstract notions, involved in the events. The enterprise resources are also entities. The state of an entity is a quantitative and qualitative evaluation of an entity at a given moment. Usually, we are interested about the state of the enterprise resources at a given moment. Given an initial state of an entity at the

moment t_1 and the events that implies the entity, between t_1 and t_2, we can determinate the final entity state at the t_2 moment.

Entities, events, actions and states have some characteristics called attributes. An attribute takes values from a predefined set, called attribute-domain. Between the concepts presented above may be defined relationships. Some of the attributes may represent references within relationships. Attributes have at their turn properties.

Making an analogy between an event description and a natural language compound sentence, we may consider an action the equivalent of each sentence from the compound sentence, the nouns may correspond to entities and determinants may be similar to the entity and action attributes.

The instances of a concept will be grouped in a collection, stored in a relational database. In the same time, the collection of the concept-instances will be managed by a corresponding user-interface. The software component refers both the persistence and the user-interface for the instances of a domain concept. The figure 1 illustrates the relationship between a domain concept and a software component.

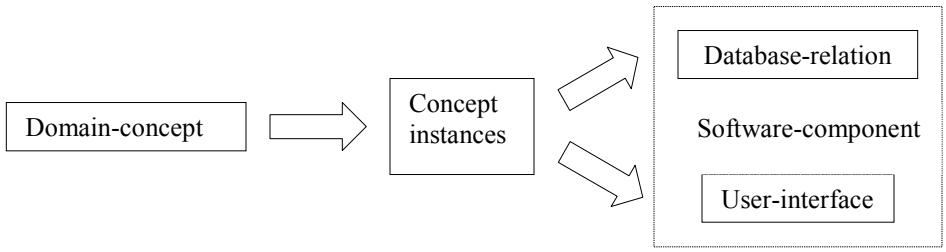

Figure 1: The relationship between a domain concept and a software component

As an example, *product* is a concept from the enterprise domain, of entity category. The *product* concept will have a corresponding *product* software component which will contain a *product* schema in a relational database and a *product* user-interface which enables browsing the enterprise products, updating them and doing all other operations required for them.

There are two types of attributes for a component: the database relation attributes which will be called *physical attributes* and the user-interface attributes, called *presentation attributes*. The set of the presentation attributes $\{A_1, A_2,\ldots, A_n\}$ for a component, must include the set of its physical attributes, $\{B_1, B_2, \ldots, B_p\}$, $p \geq n$. We'll consider, also that the component relation schema is in the third normal form. Some of the attributes will be used for the identification of the concept instance, some of them for representing instance characteristics, some for referencing other concept instances.

The component user-interface contains:

- a tabular image with the concept instances,
- a detail image with the current instance,
- a set of filtering attributes for parent-child relationships representation, and
- a set of operations required for that concept.

The component user-interface must have a data-source consisting in a database relation (the presentation attribute set is equal to the physical attribute set) or a query implying more relations from the database (the presentation attribute set includes the physical attribute set).

A CSL description defines first, the concept names and their category (event, action, entity and state). The software components will have the same name as the concepts. Additional, the component user-interface must have a data source specification: a database relation or a query.
CSL is similar to data description languages from the database management systems.

For each component corresponding to a concept, we must define a set of attributes. Each attribute has at its turn a set of properties which must be specified. The attribute properties are:

- Name — the attribute identifier.
- Alias — the attribute alias, which will label the attribute in the user-interface.
- Data type — the attribute data type (integer, long, autonumber, currency, single, double, text, date and boolean).
- Index — the name of the database index (no duplicates) the attribute belongs. If the index contains more attributes, the component attribute order will give the attribute order in the index. For the primary key index, we must specify *PrimaryKey*.
- Filter — specifies if the attribute participates, as a link element, in a parent-child relationship of the user-interface. If there are more elements in the parent-child link, the list order will be done, again by the concept attribute order. The values for this property are *Combo box* or *List box*, specifying the type of control for the parent selection.
- Size — specifies the maximum length for the attribute value (only for the text data type).
- Data source type — specifies if the attribute will be represented in the user-interface by a *TextBox* control (if the property value remain blank) or a *ComboBox* control (if the property value is *Table/Query* or *Value List*). In the case of *Table/Query* the control data source is a query and in the case of *Value List*, the data source is a set of values.
- Data source — specifies a SQL query or a set of values if the data source type property is filled.
- Required — is a boolean property which means that an attribute value must be specified, when entering a new concept instance or editing an existing one.
- Depends upon — specifies an attribute name set which determine, each of them, the value of the attribute. When an attribute from the set, changes its value, the attribute value is recalculated (when editing a concept instance).
- Read only — is a boolean property which enables the user to edit the attribute value.
- Formula — contains the expression which supply the attribute value.

The CSL descriptions may be done in specific text format and translated into a relational database format, as follows:

- Components (ComponentId, ComponentName, ConceptType)
- Attributes(AttributeId, AttributeType, AttributeName, ComponentId)
- Properties(AttributeId, PropertyName, PropertyValue)

Where ConceptType may have one of the following values: *entity*, *event*, *action* or *state*. AttributeType may be *physical* or *presentation*. We may also create user interfaces for the relational description.

Once the CSL description is achieved, we must define the automatic translation process which generates the software components: database relations and user-interfaces. The generation of the database relations is very simple, because every component associated with a concept type has a corresponding database relation with the same physical attributes as the component. From the attributes properties, only *datatype*, *index* and *size* properties are necessary.

The component user-interface generation is more difficult. The translation process takes the CSL description for a component, as input and generates the classes which implement the component user-interface. Each type of concept (entity, event, action and state) has a specific type of user-interface and for this reason a specific translation process. There are, however some common features of the user-interfaces, mentioned above (the tabular image, the detailed image, the set of filtering attributes and the set of required operations). Figure 2 shows the basic structure of a component user-interface.

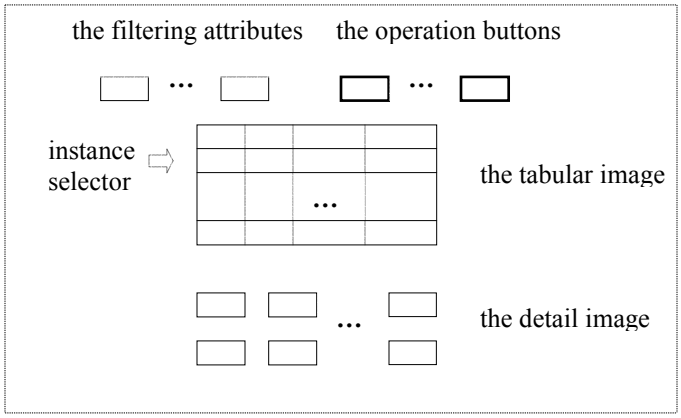

Figure 2: The structure of a user-interface component

There is also, a specific use-case for every category of concept. The translation process is guided by the following steps:

- Generate the tabular image sub-component.
- For each presentation attribute generate a corresponding user-interface control in the detail image. If the attribute is a filtering one, generates also a corresponding control for implementing a parent-child link.
- Generate command buttons controls and the corresponding event procedures for the standard operations.
- Generate private variables and methods for the user-interface.

The tabular image is a stand alone object, responsible for the concept instances browsing. There is an instance selector which determines the current instance. The current instance or a set of selected instances from the tabular image may be deleted. The detail image is used for editing a new concept instance or an existing one. Optional, the detail image presents in a single-record detailed mode, the current record. The filtering attributes implement a parent-child (one to many) relationship between the instances of two concepts (or the instances of the same concept). The tabular image contains the instances of the child concept and the filtering attribute contains the reference to the parent concept instance. The standard operations are: add a new concept instance, update an

instance, delete the current instance or the selected ones, detailed view of the current record, save changes and cancel changes. The methods implement the standard operations or general-use functions and procedures.

3. A case study of using CSL in software development

Let consider the development of a software application for a domain which uses the following concepts: product, group of products, partner, supplier/customer invoice, product reception and product stock. The concepts: product, group of products and partner are entity types. Invoice and reception are event types. Stock is a state type of the product entity. The content of the invoice and the content of the reception are action types. Each concept will have a corresponding relation in the database and a software component in the final application. As an example, let consider the description in CSL of the product and group description.

entity Group
    attribute GroupId *datatype* autonumber *index* PrimaryKey
    attribute GroupName *alias* Name *datatype* text *size* 50 *index* IGroup

entity Product
    attribute ProductId *datatype* autonumber *index* PrimaryKey *readonly* true
    attribute ProductName *alias* Name *datatype* text *size* 50 *index* IProduct
    attribute GroupId *datatype* long *index* IProduct *dependsupon* Group *readonly* true
    presentation attribute GroupName *alias* Group *datatype* text *size* 50 *filter* ListBox *datasourcetype* Table/Query
        datasource Select GroupName from Group order by GroupName
    attribute SellPrice *datatype* currency *required* false
    attribute TVA *datatype* single *datasourcetype* ValueList *datasource* 0;9;19

The automatic translation process takes as input the CSL description and generates the user-interface components and the corresponding relations in the database. Figure 3 shows the user-interface component for the *product* concept, generated from the above description. There is implemented a parent-child relationship between the group and product concepts. There is also the tabular image, presenting the instances of the product concept. On the top, there are the buttons for the predefined operations. The first button enables, if pressed, the detailed view of the current instance. The other predefined operations are settings, report printing, entering new instances and editing the current instance.

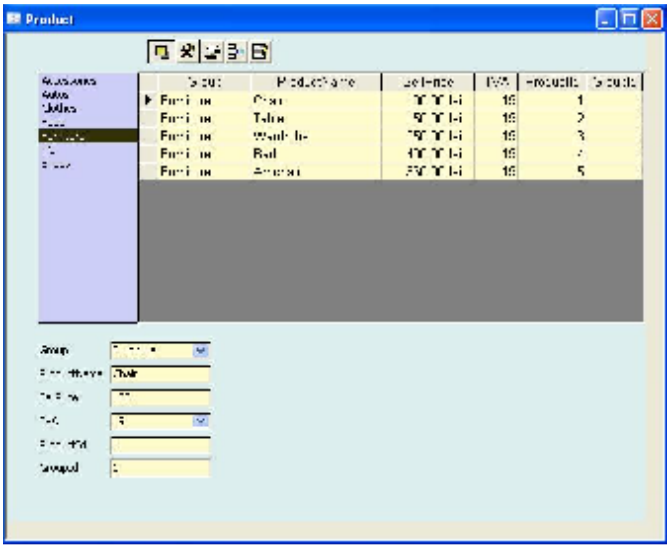

Figure 3: the user-interface component generated from the CSL description

4. Conclusions

The article presents a process of software development based on a component specification language, which starts from the definition of the domain concept. The CSL description may be automatic translated into software components. The main advantage of this process consists in minimizing the effort of the components code generation and testing, the developer being more concerned on identifying and specifying the system concepts.

The changes of the system requirements may be done more easily using this approach, because the changes affect only the specification of some concepts, the corresponding components being generated again by executing the automatic translator. Another advantage of this approach must be considered when the development tools are changed. We must rewrite only the translator.

The author developed a prototype of the translator, using the VBA language and the Microsoft Access 2003. The generated components are MS Access objects, which the developer may view and even, add new functionality or modify the contents of the generated objects. The main gain of the automatic translation is that the developer finds a lot of work already done, in general a routine work and theoretical, without errors.

5. References

[1] A.Alexandrescu, A Development Process for Enterprise Information Systems Based on Automatic Generation of the Components, *International Journal of Computers, Communications & Control* – Proceedings of the ICCCC 2008, pag.168
[2] M.Fowler, K.Scott, *UML Distilled, Second Edition A brief Guide to the Standard Object Modeling Language*, Addison Wesley, 1999
[3] R.Pressman, *Software Engineering, A Practitioner's Approach*, McGrow-Hill, 2000
[4] D.Oprea, D.Airinei, M.Fotache, *Sisteme informationale pentru afaceri*, Polirom 2002
[5] D. Zaharie, I.Rosca, *Proiectarea obiectuala a sistemelor informatice*, Dual Tech, 2003