STATEMENT OF INTEREST

CARE Technologies, S.A. develops a set of Model-Based Code Generation tools branded as OlivaNova – The Programming Machine that allow developers to automatically produce the full source code of a business application using platform-independent, conceptual models as the main development artefact.

Conceptual models are constructed in the UML-like, proprietary OlivaNova Modeling Langage and, amongst other things, they comprise four complementary views:

- Objects Model
- Functional Model
- Dynamic Model
- Presentation Model

The first three models cover the static (data and constraints) and behavioral (services and their associated functionality) aspects of an information system, while the latter is used to model presentation aspects in an abstract, platform-independent fashion.

Conceptual models can be serialized to an XML format for interchange purposes, particularly to use them as input for automatic transformation processes (both model-to-model and model-to-text).

The OlivaNova Transformation Engines produce the full source code of an information system using as input an OlivaNova conceptual model along with a set of transformation options.
Specifically, with respect to user interfaces, there are several transformation engines that target both desktop and web graphical user interfaces, using C#, ASP.NET and JSF as programming languages.
Conceptual models are platform and language-agnostic and therefore the very same model can be transformed into a C# desktop client and/or a JSF web client.

While OlivaNova Modeler allows the creation of the abstract interface models (that leverage structural and behavioral elements of other models), each OlivaNova Transformation Engine produces code in a given programming language (e.g. C#, ASP-Net, JSF) and introduces the concrete interface aspects that pertain to a certain target.

CARE Technologies, S.A. is therefore interested in attending the W3C Workshop on Future Standards for Model-Based User Interfaces in order to:

- Learn about similar approaches for the definition of abstract interfaces
- Collaborate in the definition of a common format for the definition of abstract interfaces
- Explore the possibility of leveraging task models

The part of OlivaNova Modeling Language that allows for the definition of abstract interfaces is based on a propiertary pattern language whose elements are organised in three levels:

- Hierarchical Actions Tree: that defines the access to the functionality provided by the system (a tree-like structure where leaf elements point to interaction units);
- Interaction Units: which present the user with dialogues/scenarios where the interaction with the system takes place; and
- Auxiliary Patterns: the building blocks of Interaction Units.

There are currently four types of Interaction Units:

- Service Interaction Unit, which represents the interaction of a user with the system in order to execute a service
- Instance Interaction Unit, which is an abstraction of the presentation of an instance of a model class. It comprises a DisplaySet Pattern with the observable data, an optional Navigations Pattern with the navigations to other scenarios and an optional Actions Pattern with the available actions (executions of services) on the presented instance.
- Population Interaction Unit, wich abstracts the presentation of the population of a model class. It comprises an optional set of Filter Patterns (to filter/restrict the observable population of the class), an optional set of Order Criterion Patterns (to specify the order in which the observable population will be presented) , a DisplaySet Pattern with the observable data, an optional Navigations Pattern with the navigations to other scenarios and an optional Actions Pattern with the available actions (executions of services) on each of the presented instances.Master/Detail Interaction Unit.
- Master/Detail Interaction Unit, which represents a dialogue in wich some information acting as the master data is presented along with related information acting as detail data (e.g. purchase orders and purchase order lines). It is comprised of a master interaction unit and a set of detail interaction units.

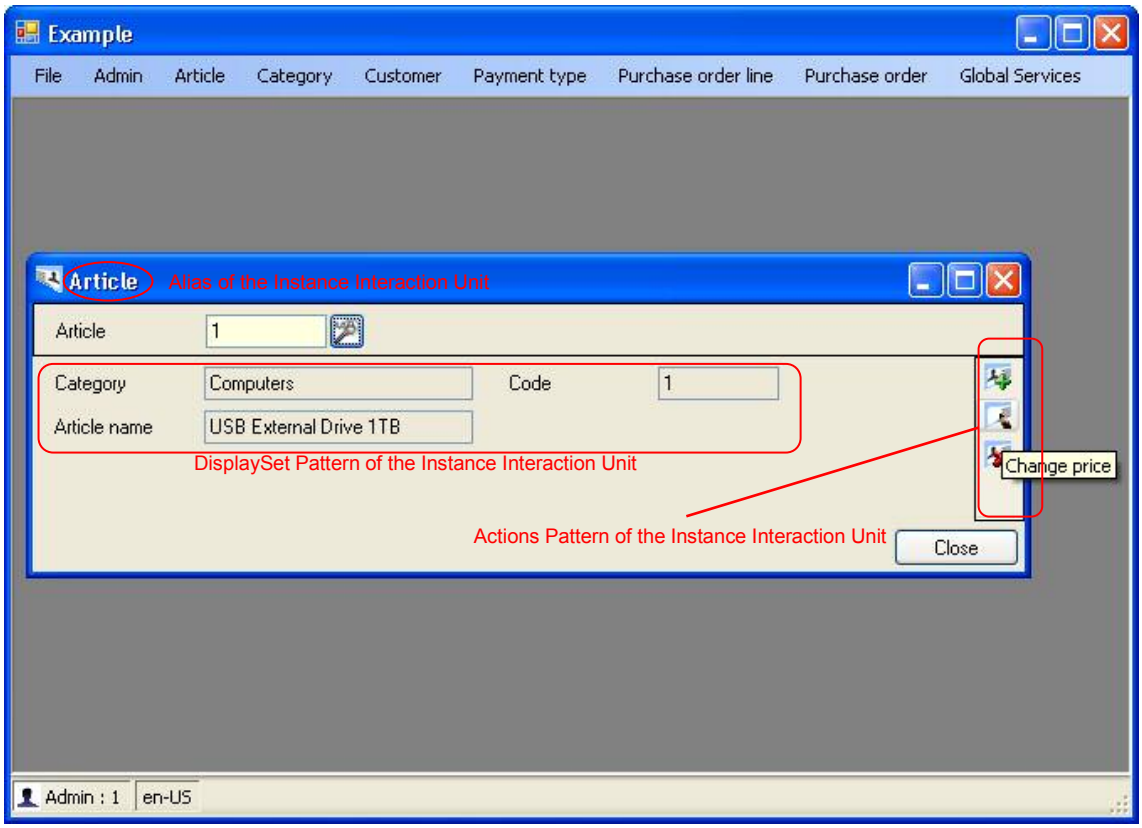
What follows is a sample XML excerpt corresponding to an Instance Interaction Unit to present an instance of class “Article”. It has a DisplaySet Pattern with three items to show the values of the name of the category associated to the article, the article code and the article name. It also has an Actions Pattern with three items to navigate to three different scenarios (Service Interaction Units actually) to execute services to create an article, change the price of an article and delete an article.

```
<! - - INSTANCE INTERACTION UNIT - - >
<PPresentacionM _Id="Clas_1UIInst_1" tipoPP="I" Nombre="IIU_Article">
        <Alias>Article</Alias>
        <Her.PPInstanciaM>
                <Ref.AccionesOfertaM _Ref="Clas_1AO_2"/>
                <Ref.ConjVisualM _Ref="Clas_1CV_2"/>
        </Her.PPInstanciaM>
</PPresentacionM>
<! - - DISPLAYSET PATTERN - - >
<ConjVisualM _Id="Clas_1CV_2" Nombre="DS_ArticleDetails">
        <ElemConjVisual _Id="Clas_1CV_2ICtjoVis_7">
                <Alias>Category</Alias>
                <Rol.PathCompleto>Category.CategoryName</Rol.PathCompleto>
                <Ref.AtributoM _Ref="Clas_2Atr_2"/>
        </ElemConjVisual>
        <ElemConjVisual _Id="Clas_1CV_2ICtjoVis_1">
                <Alias>Code</Alias> Item alias
                <Rol.PathCompleto>ArticleCode</Rol.PathCompleto> Path to attribute
                <Ref.AtributoM _Ref="Clas_1Atr_2"/> Reference to attribute
        </ElemConjVisual>
        <ElemConjVisual _Id="Clas_1CV_2ICtjoVis_2">
                <Alias>Article name</Alias>
                <Rol.PathCompleto>ArtName</Rol.PathCompleto>
                <Ref.AtributoM _Ref="Clas_1Atr_3"/>
        </ElemConjVisual>
</ConjVisualM>
<!- - ACTIONS PATTERN - - >
<AccionesOfertaM _Id="Clas_1AO_2" Nombre="A_ArticleForAdmin">
        <ElemAcOfertadas _Id="Clas_1AO_2EA_1">
                <Alias>New article</Alias>
                <Ref.PPresentacionM _Ref="Clas_1Ser_1UIServ_1"/>
        </ElemAcOfertadas>
        <ElemAcOfertadas _Id="Clas_1AO_2EA_5">
                <Alias>Change price</Alias> Item alias
                <Ref.PPresentacionM _Ref="Clas_1Ser_7UIServ_1"/> Ref to Service IU
        </ElemAcOfertadas>
        <ElemAcOfertadas _Id="Clas_1AO_2EA_3">
                <Alias>Delete article</Alias>
                <Ref.PPresentacionM _Ref="Clas_1Ser_2UIServ_1"/>
        </ElemAcOfertadas>
</AccionesOfertaM>
```

What follows is two screenshots corresponding to the above excerpted Instance Interaction Unit. The first one corresponds to a C# desktop GUI while the second corresponds to an ASP.Net web GUI.

Desktop GUI sample

Instance Interaction Unit represented as a Windows Form with its title showing the interaction unit alias.
DisplaySet Pattern represented as <Label-TextBox> pairs of widgets, with the label text showing the alias of the displayset item and the textbox text showing the value of the referenced attribute.
Actions Pattern represented as a buttons bar with different icons for each item. The alias of each item pops up as tooltip text.



Web GUI sample

Instance Interaction Unit represented as a frame of a webpage. Its alias is show both in the frame and in the browser window title.

DisplaySet pattern represented as <Text-Input> pairs of widgets, with the Text showing the alias of the displayset item and the Input text showing the value of the referenced attribute.

Actions Pattern represented as a set of hyperlinks each with the alias of the corresponding action item.