

A Model-Driven Product Line Approach for Human-Machine Interface Development

| | |
|---|---|
| <p>Marc Born, Omar Ekine <i>ikv++ technologies ag</i> <i>Dessauer Str. 28/29</i> <i>10963 Berlin</i> <i>Germany</i></p> | <p>Jon Ashley <i>DENSO Sales UK Ltd</i> <i>1 Bishop Square</i> <i>Hatfield, Hertfordshire, AL10 9NE</i> <i>United Kingdom</i></p> |
|---|---|

1. The HMI development process

In the automotive industry, suppliers are interested to re-use large parts of their implementations when delivering products to different customers. Only with a high degree of re-use and cost savings will they be able to survive in today's tough competitive environment. This holds especially for products with a relatively large part of common behavior like navigation systems. It is simply too expensive to develop the same behavior again and again for different customers and it is also not possible to address just one customer because of the high development costs which must be compensated by higher volumes.

On the other hand, the car manufacturer (OEM) has a strong interest in the realization of his own look & feel of the Human-Machine Interface (HMI) as a differentiator to other OEMs. Therefore, it is necessary to adapt the HMI to the OEM's needs and consider OEM HMI design guidelines. Tool support is definitely necessary for the development as such with advanced technologies like simulation and code generation, but also to manage the different variants a supplier provides for its customers. Both model driven development [1] and the product line approach [2] have become important paradigms in software architecture today. That is why we decided to *combine* model-driven development with a product line approach and provide tool support for *both* aspects.

2. The model-driven approach

The realization of the model-driven development of the HMI is done by the introduction of the following models:

- an abstract screen model that defines the structure of the screens of the HMI and the possible actions which can be triggered from each particular screen,
- a screen flow model (derived from UML activity diagram notation) which models the transitions between screens; the transitions originate from actions which can be triggered from the screens,
- a task model which contains tasks which are expected to be executed by the underlying core software (e.g. the navigation core).

These models are described by metamodels [3], and dedicated graphical editors supporting the notation of these models are implemented in the Eclipse environment [4], [5]. In addition to these models, there is tool support available to define the look and feel of screens (the so called skins). The skin definitions can be mapped to the abstract screen model.

In order to fulfil the requirement for rapid development and improve communication with customers, screen flows may be simulated (**Figure 1**). Simulation means to execute the screen flow model according to the execution semantics of UML activities.

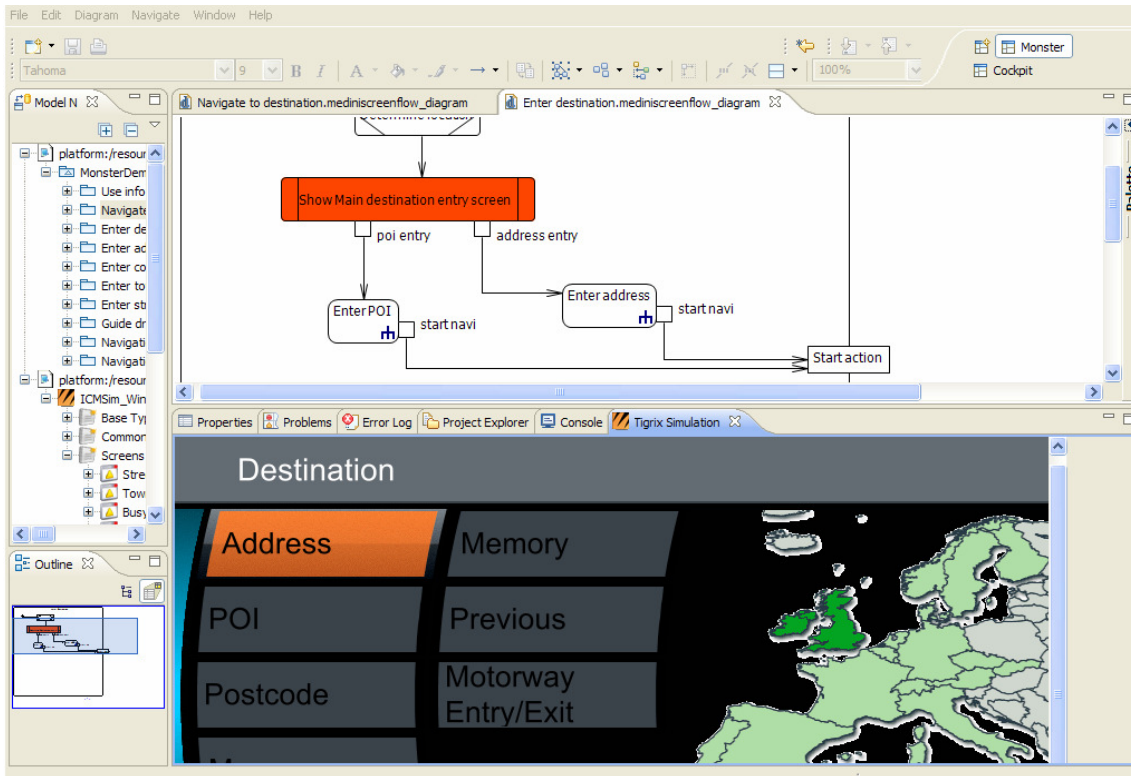


Figure 1: Simulation of screen flow model

To trigger actions that result in screen transitions during a simulation, several user interfaces are provided. The user can either click on the pins of the activity or action in the screen flow model or he can use the real HMI skin. Then, the simulator uses the mapping from the skin to the abstract screen model which is to be provided beforehand. The simulator is flexible in that respect, so other sources of action triggers are also possible and can be integrated.

3. Product line support

Model-driven development and simulation capability already offer a significant productivity enhancement. In addition we decided to apply a product line approach for variant support and extended the modelling capabilities to express variability in a so called *family model*.

To specify variability in the screen flow model we need to express at least:

1. Screen flows which are present in some variant models and not present in others (*optional flows*)
2. Invoked screen flows which are differently realized in some variants (*variant flows*).

For that purpose, a new modelling concept is included in the screen flow model: at so-called *variation points* [6] which are inserted in a transition, alternative screen flows can be invoked. An example is illustrated in **Figure 2**. In this example, we want to model the fact that a different “settings” screen flow is used for variants of the HMI that are delivered to different markets. In the family model, the `settings` action has an outgoing transition leading to a variation point (notated with a circle). From the variation point different screen flows (Asia market, US market, Europe market) are the target of the transition.

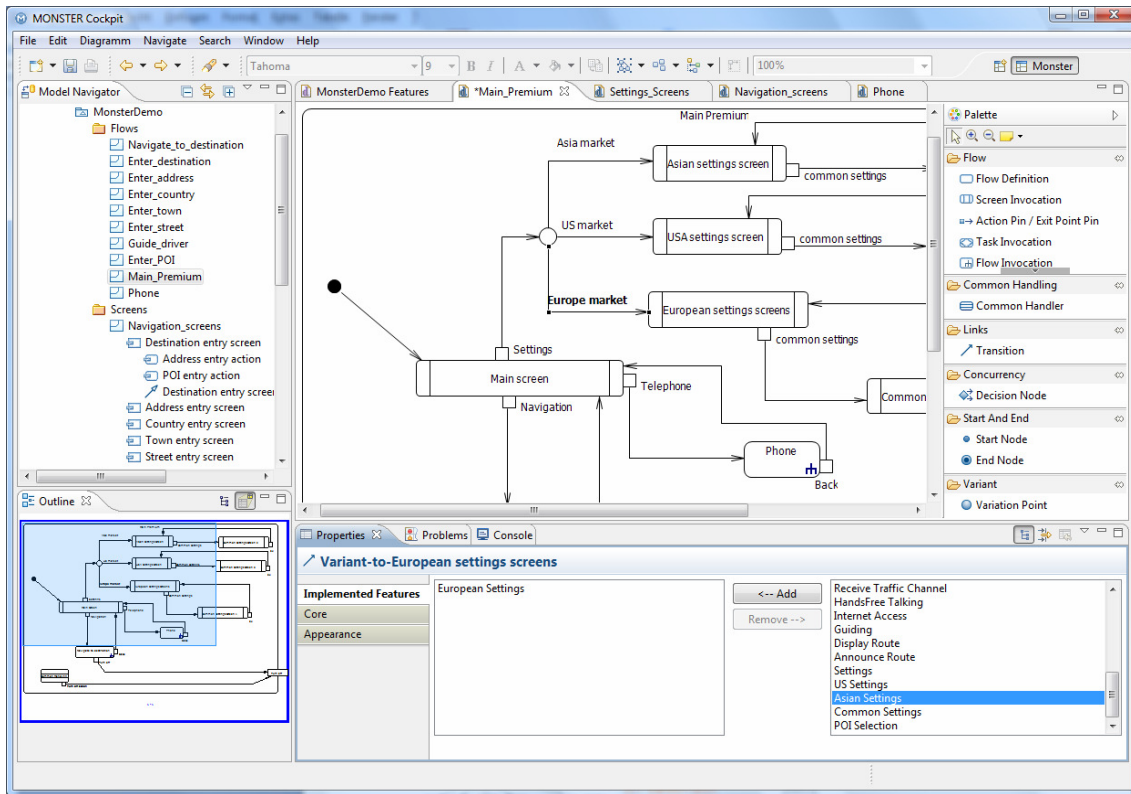


Figure 2: Mapping features to screen flows in family models

Assuming that we have the possibility to define a family screen flow model which contains all possible variants, we can use *feature models* [7] to derive concrete variant screen flow models out of such a family screen flow model. Based on the feature model, the selection of the variants to be applied for a certain product is made.

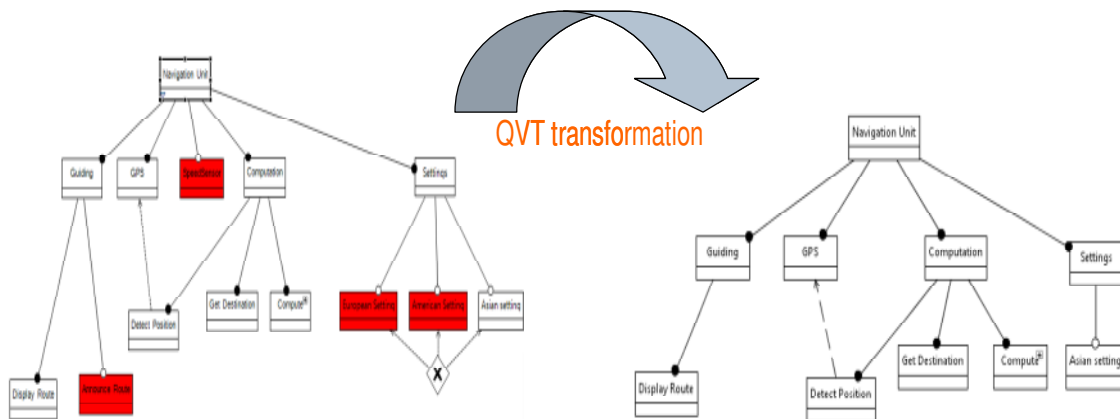


Figure 3: Variant feature model derivation

To derive the variant screen flow model out of the family screen flow model, we applied two model transformations using our transformation engine based on QVT [8]. First, it derives the product feature model out of the family feature model based on the feature selection (**Figure 3**). Then, it uses the mapping of the features to screen flow definitions (**Figure 2**) to decide for each variation point which of the flow invocations has to be included in the variant screen flow model. (The first transformation is a reduction of the feature model.) After the transformation process, the derived variant screen flow model (**Figure 4**) is validated to assure that a valid variant model is always generated.

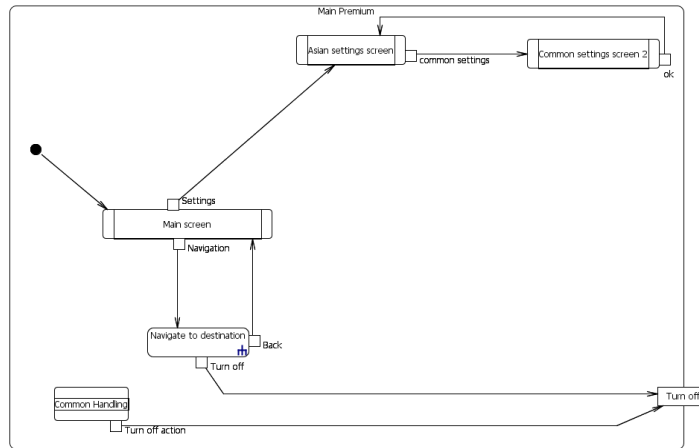


Figure 4: Product Screen Flow Model

4. Conclusions

With the approach described herein, we have shown that model-driven development and the product line approach can be combined in a useful way to take advantage of both approaches – the productivity advantage can be realized due to the integration of both approaches: flexibility and automation as well as a high degree of reuse. The usage of the QVT relational engine for variant model derivation offers the possibility to benefit from the backward transformation capability of QVT. When a variant model is adapted or extended, and it is obvious, that such extensions or adaptations are useful for other customers, they can be backward-propagated into the family model without extra effort.

Our next step is to apply the approach in multiple DENSO customer contexts in order to measure efficiency improvements as well as the impact of each project carried out on the expected effort for subsequent projects. Furthermore, we will extend the variant generation to also cover the connection to the navigation core API.

5. Demonstration Scenario

The demonstration will show an example family model of a car navigation HMI. The family model will contain the abstract screen model, the screen flow model and the family feature model. Furthermore, concrete skins are available. It is demonstrated, how the model can be simulated. Based on different feature selections, variant models are generated automatically and can also be simulated.

6. References

- [1] Object Management Group, Model Driven Architecture (MDA), ormsc/01-07-01, July 2001.
- [2] Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures. Addison-Wesley, Boston (2004)
- [3] IKV⁺⁺ Technologies AG, medini meta modeller, <http://www.ikv.de/medini>
- [4] Eclipse.org, Eclipse Modeling Framework, http://www.eclipse.org/projects/project_summary.php?projectid=modeling.emf
- [5] Eclipse.org, Graphical Modeling Framework, <http://www.eclipse.org/modeling/gmf/>
- [6] Jacobson, M., Griss, M., Jonsson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison-Wesley-Longman, Menlo Park (1997)
- [7] Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute, Carnegie Mellon University, Pittsburgh (1990)
- [8] Object Management Group, MOF 2.0 Query / Views / Transformations RFP, OMG document ad/02-04-10, April 2002.