

A Flow Centric Interaction Model for Requirements Specification and User Interface Generation

Thomas Schlegel^a, Alexander Burst^b, Thomas Ertl^c

^a Fraunhofer Institute for Industrial Engineering (IAO), Stuttgart, Germany.

^b ETAS Engineering Tools GmbH, Stuttgart, Germany.

^c Visualization and Interactive Systems Group (VIS), University of Stuttgart, Germany.

Abstract. A huge amount of information from customers, users and other stakeholders is present in the early phases of interactive software development. To preserve and capitalize this knowledge, a descriptive and flow-oriented Interaction Modeling Language (IML) is presented that is based on XML and Use-Cases. The ability to use this specification for user interface prototype generation renders the approach directly applicable for user-oriented rapid software development projects in practice. An abstract user interface representation and a language for the description of code generation rules complete the toolset and allow the embedding into a more generalized Soft Artifact Engineering approach.

Keywords. Interaction Specification, User Interface Generation, Interaction Modeling Language (IML), Soft Artifact Engineering (SAE)

1. Introduction

The early phases in a development project for interactive software are characterized by the presence of a huge amount of information from customers, users and application domain experts. Although this knowledge is critical for the design of an application and especially a user interface that meets customer and user needs, it is often not preserved or even lost in the subsequent steps of the project. This critical preservation of knowledge available in the early phases can only be accomplished using integrated models that stay consistent and up-to-date over the whole user interface development process and contain all necessary information for the development of the specific user interface.

Approaches have already been suggested using different types of models such as task models, domain models and presentation models like in Teallach (Barclay et al. 1999) or OOA-Models like in JANUS (Balzert 1994). These types of models are intended for utilization in user interface development environments or even user interface generators. Being highly developed, most of these models lack simplicity – in the meaning of easy creation – and often have only rare connection to other models in the software development process.

For broad acceptance in software development practice, models have to integrate seamlessly into one or more of the common models created, like UML Use-Case-Diagrams and descriptions, Class-Diagrams, etc. (OMG 2003) (Henderson-Sellers et al. 1999). As such description techniques are broadly used, enhancements to these models show easier acceptance by developers.

While functional and technical aspects were the main aspects in the past, nowadays interaction and non-functional

requirements have gained high importance in conjunction with topics like software quality and usability. This requires a stronger involvement of non-developers and the adoption of models for more “soft” aspects like user interfaces. Additionally, applied software development processes have shifted from straight forward models like waterfall to iterative, incremental and sometimes agile approaches to allow back coupling (Pressman 1997).

Prototyping has become popular and leads to better integration of customers and users, transforming formerly incomprehensible requirements into visible and understandable artifacts that can serve as a basis for further discussion in order to integrate all stakeholders and achieve a win-win-situation. Building these models, software developers often encounter the dilemma of either spending too much effort in prototype development or not considering user and customer requirements as much as they deserve.

2. Modeling and prototyping in one close loop

A solution to this problem is the integration of requirements specification with interaction models and prototype generation in one close loop. This direct dependency and back-coupling ensures consistency between these three aspects: Requirement models are forced to include information about interaction and can be directly transferred into pure interaction models or even a user interface prototype. On the other hand, the discussion of the generated UI prototype directly reveals shortcomings and necessary changes in the specification.

To achieve this, an integrative model is needed that contains all necessary information but also follows description and perspective of user and customer. There are already many examples for user interface generators that bring appropriate

models with them, e.g. (Janssen et al. 1993) and (Ziegler 1996), and interaction oriented description methods, e.g. UMLi (Pinheiro da Silva 2000). The problem with these rather complex and often data-centered or object-oriented models in UI generation – e.g. OOA (Hofmann 1998) – is that users and customers usually require and accept only models that are easy to understand and represent their workflows and view of the system. Developers on the other hand may accept such models but will not invest the effort for the creation of models that are not able to save enough effort in later phases or are not at least effort-neutral in practice.

In the following we describe a solution that employs the enrichment of a common model with additional information and the construction of a model-based multi-step, multi-target generator for user interface prototypes.

3. Selection and extension of the model

Over the last years, UML Use-Cases have become a very popular method for descriptions in the analysis and specification tasks of software development projects. Easy graphic modeling and free description of workflows for each Use-Case allow for modeling and understanding by non-developers, which has been the basis for their success in practice. The advantages of ease and flexibility for the early stages become a problem when these specifications are to be used in design and development – especially in user interface prototype generation.

Therefore, we propose a model that includes mechanisms for structuring and detailing the initial Use-Cases with regard to user interaction. This will help to preserve all information provided in the basic Use-Case and allow for discussion of the model. But it will also give the developer the ability to use this extended Use-Case model in the complete development cycle, providing a consistent requirements and interaction model and the ability of out-of-the-box user interface generation.

4. Interaction Modeling Language (IML)

We call this XML-based extension of the Use-Case paradigm Interaction Modeling Language (IML). Figure 1 shows a condensed example view of an IML-Model containing three parts: Project definition, data definition and only one IML Use-Case.

4.1. IML Project Definition

In this XML structure view, the hierarchical construction of IML becomes visible. It provides a project definition that embodies necessary information for handling and generation, including stakeholders and natural languages used. The natural language support allows for easy localization giving the generator engine the ability to check model completeness for each destination language. This becomes possible because all translations are stored with their interaction context allowing easy access of the semantics.

4.2. IML Data Definition

The data definition is initially empty and is completed when data sources and sinks are needed while performing the incremental modeling. It can also be completed during the

Use-Case model refinement and transition to design. The defined data pools serve as a connecting basis for the interaction specification. For the generation of a complete prototype, it is even possible to define database access at this point.

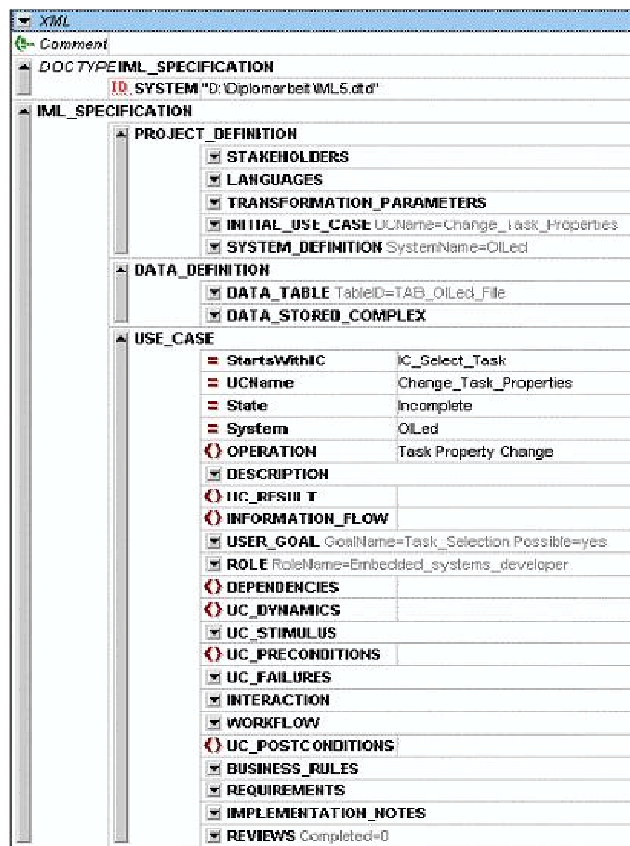


Figure 1: Sample IML model (XML-based)

4.3. IML Use-Case Definition

The third part is the real Use-Case part. According to structures as proposed e.g. in (Ambler 2000), a Use-Case contains facultative and obligatory parts that are to be completed during the specification. The elements defined here are a result from necessities in practice and model requirements.

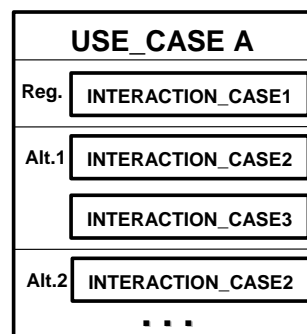


Figure 2: Use-Case structured with regular and alternative flows.

With many facultative parts and wildcard items, IML allows for an incremental development. As the focus for this work

lies in the interaction description, requirements and consistency elements are not discussed in detail. IML Use-Cases are structured hierarchically, while Standard Use-Cases describe sequences line by line. These workflows described textually in a basic Use-Case are included and structured in the interaction part of the above IML Use-Case definition.

Simple sentences have to be transformed to an XML structure preserving and expanding the information already contained in the standard Use-Case description. Each Use-Case owns the elements or parts shown in figure 1 and 2. To identify the usual workflow of a Use-Case, the interaction part is divided into a regular flow and a set of alternative flows.

While the structure of both is similar, the regular flow represents the entry point for the user interface generator to start with. Actions in the alternative flows are referenced from the regular flow.

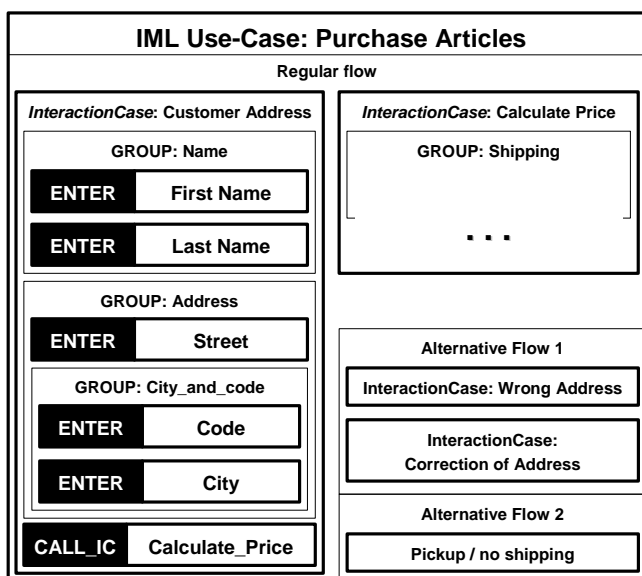


Figure 3: Sample structure of an IML Use-Case definition.

5. InteractionCases

So far the IML concept mainly complies with the regular Use-Cases already used in practice, indeed structuring them more consequently and allowing or forcing the presence of additional information.

Textual descriptions in UML Use-Cases usually include a variety of different steps that are necessary to accomplish a Use-Case. But often these steps do not share a common topic from the perspective of an interaction designer, because they always traverse a complete process or provide problem solutions from the application point of view. The key approach for structuring and refining basic Use-Cases is therefore to subdivide them into more atomic components using the concept of InteractionCases, which we have developed extending the Use-Case paradigm.

We define an InteractionCase as an interaction sequence that is strongly independent from other sequences and consists of one or more system or user actions that are part of the same Use-Case. These actions form a procedure that should not be

split up further in order to concentrate all information necessary for an interaction but does also not contain steps that belong to a different context. This concept transfers the principle of strong internal cohesion and loose coupling with other components to interaction specification.

For example, in an order process, the entry of the address for dispatch could be an InteractionCase, because it has only loose coupling to the price calculation step but high cohesion in its content. For this reason, an InteractionCase contains all steps that belong together as one interaction flow.

As shown in figure 3, the steps necessary to enter a complete address form one InteractionCase, whereas “Calculate Price” is a different task – and probably also a different screen in the final user interface. Therefore, the workflow for price calculation is being separated from the address entry interaction flow by encapsulating it into a second InteractionCase.

Other InteractionCases error handling and more unlikely selections are located in the alternative interaction flow sections attached to the common Use-Case “Purchase Articles”. The different atomic interactions like entry of first and last name can be additionally semantic-grouped within their InteractionCase.

It becomes easier to define what should be the size of an InteractionCase, if it is considered that the translation of an InteractionCase into a graphical user interface will often be a dialog or tab.

An InteractionCase covers an entity that has the same topic and contains interrelated information and interaction elements. The amount of interaction steps is similar to a dialog window in a graphical user interface or a dialog sequence in a speech user interface.

6. Structuring InteractionCases

InteractionCases are networked with each other by different flow relations, which make it possible to switch to other InteractionCases or even other Use-Cases depending on various conditions. Like modules in software engineering contain classes, which again contain elements, InteractionCases contain different interaction steps like enter, select, edit and write (see figure 4). These elements can be used directly or recursively grouped by semantic group elements that structure the interaction flow. Three of the basic elements are described in the following.

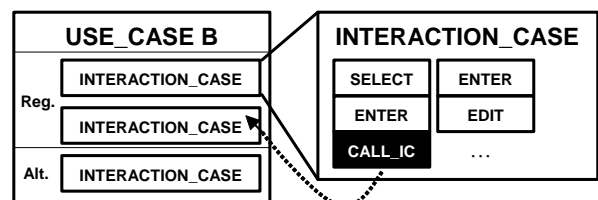


Figure 4: Structure of Use-Cases and InteractionCases

6.1. Enter Actions

Enter actions define interactions that require the input of new information by the user. Though standard value suggestions

may exist, the system holds no representation of the information space the information will come from.

6.2. Edit Actions

Edit actions allow for display and manipulation of information already existing in the system. The user is not required to enter information but may change the existing information if appropriate.

6.3. Select Actions

Select actions let the user chose a subset of elements in a pool. While the elements are known, the user needs to select which of them are appropriate in this context. There also exist hybrid forms like selections with item pools expandable through enter-like actions.

6.4. Sub-Structuring with IML Groups

Often a further substructuring mechanism is needed to create blocks inside an InteractionCase, which emphasize the higher coherence of e.g. first name and last name compared to first name and street in an address. The group element allows a hierarchical structuring of elements as well as groups and provides description mechanisms that help the user interface designer or generator decide how to transform the grouping meta-information into artifacts and rules for user interaction and UI structures.

Grouping meta-information can be used to prevent separation of elements, to shift elements to a sub-dialog or to arrange elements following the law of proximity. Also calculation of metrics like elements per group or InteractionCase is facilitated and helps making suggestions for dialog structure improvement already before dialogs are finally created.

7. Towards an Abstract Graphical User Interface Model

Although an IML model sticks close to the user perspective and workflows, it contains enough information to transform the specification into a generic UI model that can serve as a basis not only for dialog design but for user interface prototype code generation. In connection with IML, we have developed a simply structured representation for a generic user interface description – the Dialog Layer Language (DiLL). This intermediate language is necessary to allow manual changes in the generated user interface definition without manipulation of code fragments and resource definitions.

DiLL abstracts changes by UI developers from the code layer and provides back links to the IML model for operations that require IML model information. A DiLL model is created automatically from a valid IML model by the transformer.

The Dialog Layer Language is intended only for graphical and textual user interfaces as it deals with screens, groups and different types of elements. For other interaction technologies like speech, a different intermediate representation will be required.

8. User Interface Prototype Generation Process

The user interface generation process follows a pipeline approach to allow linear transformation and accomplish the

different tasks from model completion to code generation. The process steps are defined as follows.

8.1. Completion

Usually, an IML model is not fully completed at generation time due to incremental development or it needs completion steps – like Help IDs – that can be accomplished automatically. Therefore, the completion step checks the IML source model for consistency and completes missing entries as far as possible. If severe problems occur, the transformation process is interrupted.

8.2. Transformation

The transformation step interprets the IML model and forms a DiLL model on the basis of the information given in the IML model. The DiLL model contains all interaction elements that are necessary to complete the tasks defined in the IML model: A generic user interface has been created on the top of the IML specification.

8.3. Optimization

Although the DiLL model contains the necessary interaction elements and the overall structure, several optimizations are possible. Balancing dialog elements, separating defined group types from the dialog and the calculation of dialog metrics form only some of the functions that can be applied. Applied design rules can range from simple ones like the “magical number 7 plus or minus 2” (Miller 1957) per group or window to complex calculations and conditions depending on field types and their translation to composite elements.

8.4. Generation

The optimized DiLL model is the ideal basis for the code generation. For each target a unique Element Transformation Description (ETD) file exists that described the DiLL to code transformation. This generation target is characterized by the destination platform and the destination programming environment used by the software developers dealing with the results of generation.

The ETD format used for the code insertion is an XML based notation for the code to be generated per specific element type. The generator uses project templates that contain the basic project files together with code insertion marks for the desired platform. Recursive and iterative constructs, counters and conditions form the basic elements in ETD for the insertion process.

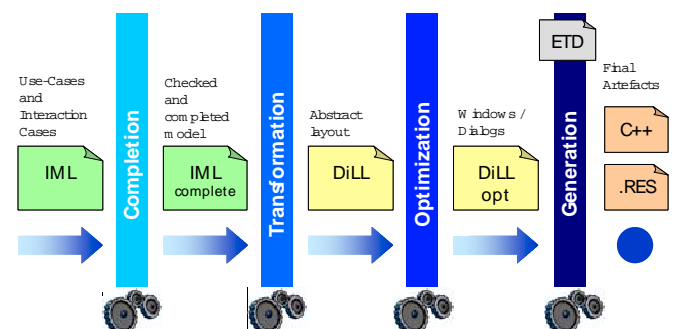


Figure 5: Transformation Process

The user interface generation follows the process shown in the figure 5. It is very similar to the visualization pipeline in computer visualization.

9. Generation of other final artifacts

So far, we have only discussed the ability to generate user interface prototypes based on an IML models. But a fully developed IML model provides enough information to allow for generation of artifacts beyond GUI models.

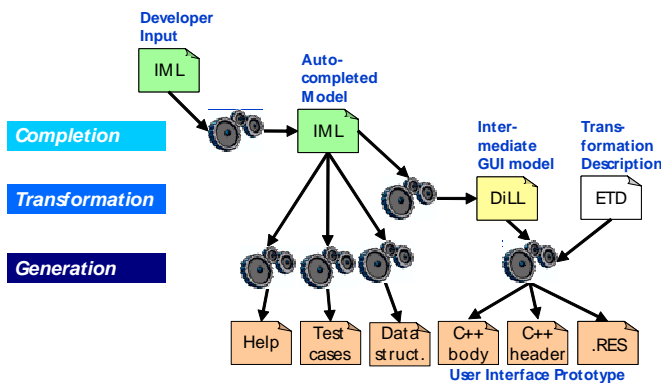


Figure 6: The generation process also provides final artifacts different from generation results.

If the textual description sections in the IML models are completed, it is possible to generate help files and user documentation from the IML model. Once the IML model has been completed, black box test cases for the application testers can be generated from the information given in the interaction flows and attached data definitions.

The advantage of these and other generatable artifacts is that changes in the IML model have direct impact on them. Generating artifacts from an IML model, it is not necessary to find and track changes on every artifact but only on the IML model. Using the IML approach for generation of the artifacts as shown in figure 6 will allow for application of IML as an integrated base model for user centered development processes. Documentation, including user manuals and development status reports can then be generated from appropriate information given in the IML model.

10. Application of the approach

To verify the practicability, a transformer prototype for user interface generation has been developed that realizes a subset of the concepts defined in (Schlegel 2002). This prototype reproduces the complete pipeline described above and generates a Visual Studio C++ project that contains all files necessary for direct compilation and further work with this project.

Using IML for description and generation of a basic UI prototype for configurator software of an automotive tool suite at ETAS GmbH, the concept has proven its strengths especially in complex configuration workflows that require a flexible description of interaction scopes and dependencies between interaction elements (Schlegel 2003).

First tests of IML at ETAS have proven that the semantic structuring of Use-Cases is improved and that software developers show an increased awareness of Usability issues especially at the early stages of software development projects. Especially, the ability to integrate natural language translation directly into the model improves software quality when a localized variant of the software has to be build. As of the generator's prototype state, it has not been possible to gather reliable results on its application in practice.

As IML focuses on ramified interaction sequences with clearly described atomic interactions, it is quite poor for construction tools like CAD but strong for configurators and other complex data entry and manipulation applications. Further research and development will have to be carried out regarding the formation of a consistent meta-model for all models and an appropriate visualization and editing approach as well as restructuring of transformation and description rules.

11. Outlook / Embedding into SAE

To enhance the modeling capabilities and to further integrate the interaction specification into more technical or organizational development processes, the approach presented here will be integrated into the Soft Artifact Engineering (SAE) approach.

SAE is a holistic and integrative approach currently developed at Fraunhofer IAO to allow the integration of different stakeholders like managers, usability experts, and software developers into development processes for virtual products – i.e. “soft artifacts”. The goal is to integrate all models into one component-based and object-oriented meta-model that allows for interconnecting completely different models like interaction specification and business process models. A high degree of freedom for modeling and interconnecting models will be the advantages of these efforts on integration and will lead to an integrated interactive systems development approach supported by user interface prototype generation.

12. Acknowledgements

The authors want to thank ETAS GmbH and the VIS, University of Stuttgart, for their kind support of this work. Many thanks for his great support of the work on this project go to Dr. Matthias Ressel. Ongoing application and integration of this research to the service engineering domain is conducted in the LIKE project with support of the German Ministry of Education and Research (BMBF).

13. References

- Ambler, S.W. (2000): Documenting a use case: What to include, and why. Retrieved March 19, 2004 from <http://www-106.ibm.com/developerworks/webservices/library/ws-tip-docusecase.html>
- Balzert, H. (1994): Das JANUS-System: Automatisierte, wissensbasierte Generierung von Mensch-Computer-Schnittstellen. In: Informatik - Forschung und Entwicklung 9, 22-35 (1994), Heidelberg: Springer Verlag.

- Barclay, P.; Griffiths, T.; McKirdy, J.; Paton, N.W.; Cooper, R.; Kennedy, J. (1999): The Teallach Tool: Using Models for Flexible User Interface Design. In: Vanderdonckt, J.; Puerta, A. (Eds.): Proceedings of Computer-Aided Design of User Interfaces II (CADUI'99) (pp. 139-158), Kluwer.
- Henderson-Sellers, B.; Firesmith, D.G. (1999): Comparing OPEN and UML: the two third-generation OO development approaches. In: Information and Software Technology 41.3 (pp. 139–156).
- Hofmann, F. (1998): Grafische Benutzungsoberflächen: Generierung aus OOA-Modellen. Heidelberg, Berlin: Spektrum, Akademischer Verlag.
- Janssen, C.; Weisbecker, A.; Ziegler, J. (1993): Generating User Interfaces from Data Models and Dialogue Net Specifications. In: Proceedings of INTERCHI'93 (pp. 418-423), ACM Press.
- Miller, G. A. (1957): The magical number 7 plus or minus two: some limits in our capacity for processing information, Psychological Review Vol. 101, No. 2 (pp. 343-352).
- OMG (2003): UML 2.0 Superstructure Specification, OMG Final Adopted Specification, Retrieved December 19, 2003, from <http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.pdf>
- Pinheiro da Silva, P.; Paton, N. W. (2000): UMLi: The Unified Modeling Language for Interactive Applications. In: Evans, A.; Kent, S.; Selic, B. (Eds.): UML 2000 - The Unified Modeling Language. Advancing the Standard (117-132), Springer Verlag.
- Pressman, R. S. (1997): Software Engineering – a Practitioners's Approach, European adaptation, Fourth Edition. New York: McGraw Hill.
- Schlegel, T. (2002): Entwurf und Erprobung eines software-gestützten Verfahrens zur Anwendung software-ergonomischer Methoden in den frühen Phasen der Anwendungsentwicklung. University of Stuttgart / ETAS GmbH Stuttgart.
- Schlegel, T.; Burst, A. (2003): Interaktionsmodelle in der Spezifikation: Von der Datenzentrierung zur Ablauforientierung. In: Ziegler, J. (Hrsg.): i-com, Zeitschrift für interaktive und kooperative Medien (pp. 17-27), Oldenburg Verlag.
- Ziegler, J. (1996): Eine Vorgehensweise zum objektorientierten Entwurf graphisch-interaktiver Informationssysteme. Berlin: Springer Verlag.