Model-based User Interface Generation from Process-Oriented Models

Thomas Schlegel

*Institute for Visualization and Interactive Systems, Universität Stuttgart
Germany*

1. Introduction

Usability and software ergonomics play an important role in software product quality and success today. While software engineering has lead to structured approaches and processes for improving quality, usability issues and user interface design are often still regarded as rather optional or late-phase activities. A reason for this gap is that user interface design is either executed ad-hoc and unmanaged by software developers during application development or as a separate activity with only loose coupling to other elements of the software development process.

Already in the late 1980 models and systems for user interface design support have been developed by researchers, followed by first interpreter-based or generative approaches for transforming interaction-oriented descriptive models into user interface prototypes or even full applications. Different types of models or even model sets were created to provide enough semantics for user interface generation.

While generative approaches in software engineering like OMG's Model-Driven Architecture (MDA) were comparably successful in practice also, dedicated modeling and generation approaches for user interfaces were not as successful. The problem of most approaches is that complex and separate models for the user interface only have to be created, which do not integrate well with the software engineering processes and models. Especially where additional effort has to be spent and models only benefit the dialog layer, acceptance by software developers and even potential users has been low.

Especially the development of interactive software has proven to require the involvement of software engineering methods as well as usability aspects. To give emphasis to quality aspects – especially usability – in software development projects, different methodologies have been invented, ranging from checklists to dedicated modeling approaches, complete User Interface Development Environments (UIDE) and User Interface (UI) generators. Approaches trying to focus on usability issues only for the most part have not had much impact on practice and further research because of their incompatibility to existing software engineering models and processes that are used in software development.

A key factor here is the integration of concepts, perspectives, processes and models of usability experts, software engineers and other project-critical stakeholders (Alexander & Robertson, 2004) but still allow for early-phase models and prototyping approaches that

have proven their strength in interactive system development. The communication between stakeholders as well as the creation of artifacts is and will be based on the successful use of common models shared when developing software that provides a user interface.

As a consequence, this chapter presents an integrated modeling approach that allows to start with basic specifications such as essential use cases followed by an incremental and iterative refinement and enhancement towards an extensive specification of interaction and system reactions. The fundamentals of this Interaction Modeling Language (IML) were developed in conjunction with a user interface generator approach for the creation of graphical user interface prototypes based on a flow-centric interaction specification using IML. This suitability for automatic processing eases the deduction of interactive prototypes from user-oriented models already in early phases of software development projects.

2. Existing Modeling Approaches

An extensive model landscape has been developed in the last decades for analysis and design in software development with regards to user-oriented approaches. This section provides an overview necessary to understand the current model and model integration discussion.

2.1 HCI Models for Task Analysis and Modeling

The goal of finding a modeling technique that serves for task analysis and modeling has produced a variety of different techniques in the human-computer interaction domain, ranging from psychological to rather technical methods.

Some of them are described in the following to give an impression of the wide range and provide hints for developers and research projects that may have special requirements pointing to one of those methods.

ETIT analysis assumes that users have to map intern tasks (IT) to extern tasks (ET). Using a matrix it is possible to estimate the effort for knowledge transfer for similar tasks.

GOMS (goals, operators, methods, selection rules) takes over the notion of goal, sub-goal, operator method and selection rule from the problem solving theory. Goals are subdivided into sub-goals; activities arise from the application of operators, which transform to methods – i.e. high-order operators – by control structures.

CLG (common language grammars) are better suited for command based systems. It differentiates a task layer with extern tasks, a semantic layer with intern tasks, a syntactic layer with concepts and commands and an interaction layer with actions and presentation.

TAGs (task action grammars) do not have this focus on commands, which makes them suitable for other user interfaces, too. They provide a meta language for the description of task description languages. A tag model contains task elements (features), a list of tasks and dedicated elements and substitution rules (grammar) for the production of activities.

CCT (cognitive complexity theory) helps determining the cognitive complexity of a system by judging the knowledge necessary. CCT has been developed from GOMS by projection of the procedural descriptions to conditional and action rules.

Task cases or essential use cases describe the interaction of user and system in natural language with only little structure like dialog or card form. (Constantine, 1995)

These models target behavioral modeling, which is more or less procedural, and often have no relationship to software engineering models. As most of these models are completely

incompatible with modern object-oriented software development approaches, they are often used by a small group of specialists only, which makes the models only rarely useful as a model basis for software development projects and integration with software engineering. With the rise of UML (OMG, 2007; OMG, 2007b) use cases in a combined graphical and textual form have been introduced to software engineering practice as black box modeling concepts.

2.2 Models for Task Sequencing

Describing sequences has also lead to many different models trying to capture the interaction dynamics. Many of these are based on concepts similar to Petri nets (Reisig, 1986):

- Data-Flow Diagrams (DFD; DeMarco, 1979)
- State-Charts (Harel, 1987)
- Event Schemata (Martin & Odell, 1992)
- EPC (Scheer, 1994)
- UML sequence diagrams (OMG, 2003)

Others have been described by (Larson, 1992) and (Janssen et al., 1993). They have often been used as the dynamics description part of models for user interface generation and have also had success as partial interaction models.

2.3 Object-Oriented Methods

With object-oriented paradigms becoming popular in the field of programming languages under the notion of object-oriented programming (OOP), modeling concepts also have turned in this direction to keep paradigms of models and code compatible.

In the early nineties, for example OBA (Rubin & Goldberg, 1992), OOA (Coad & Yourdon, 1991) and OOA&D (Martin & Odell, 1992) have been published, all aiming at an object-oriented analysis and providing a basis for the following object-oriented design. (Janssen et al., 1993)

The concepts of OMT (Rumbaugh et al., 1991) and OOSE (Jacobson et al., 1992) were integrated with the approach of Booch to form today's industrial quasi-standard Unified Modeling Language (UML; Scott, 2004) for object-oriented analysis and design, while the OPEN modeling language (OML; Firesmith et al., 1998) has been developed as a sort of competitive language to UML. A comparison of both languages from the perspective of OML can be found in (Henderson-Sellers & Firesmith, 1999), where it is argued that UML can only be considered as hybrid object-oriented and is strongly focused on C++ implementations.

Although, with the Meta Object Facility (MOF; OMG, 2002), OMG strongly pushes UML into the direction of model integration, which will lay the fundament for emerging integrated modeling approaches like the one presented in this chapter.

With the introduction of semantic web technologies, the semantics of models and their (semi-)automated interpretation / inference have become more important leading to more powerful models, which still focus mainly on structural aspects like data structures, types and dependencies.

2.4 Process-oriented Approaches

Besides modeling approaches, many forms of user-oriented software development methodologies have also been discussed.

One popular example is user-centered design, which is an approach for integrating the user into development of software and in newer publications also to hardware and services (Vredenburg et al., 2002), for example using a User Centered Design Process (ISO, 1999). The User Centered Design Process involves users in an iterative process. It strongly involves early evaluation of the user interface, for example using paper prototypes, to gain user feedback as fast as possible and let users actively participate in the development activities. Research on possibilities for integration with software engineering has also been carried out (Metzker & Seffah, 2003).

Usage-centered design on the other hand has emerged from a software engineering perspective and therefore aims at integrating user-oriented models like task descriptions into model-based software development processes ranging from very structured to agile methods like extreme programming (XP; Beck, 1999). Instead of user testing, the focus here lies on improving usage based on models. A short comparison of user centered design and usage-centered design can be found in (Constantine et al., 2003; Constantine & Lockwood, 2002).

2.5 Model-based Development

The problem with these rather complex, often data-centered or object-oriented models in UI generation – e.g. OOA (Hofmann, 1998) – is that users and customers usually only request and accept models that are easy to understand and support 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.

Recognizing this problem and the necessity of an integration of usability and interaction aspects into existing software engineering models, different interaction-oriented description concepts have been developed, improving and extending existing software engineering modeling languages like UML. One example for such an extension is UMLi (Pinheiro da Silva, 2000), an UML extension for interaction design. This extension of common concepts allows easier roll-out than the dedicated models described above. Unfortunately, generative processing is not possible with most of them due to missing formalization.

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 & Firesmith, 1999). As such description techniques are broadly used, enhancements to these models show easier acceptance by developers.

While these functional and technical aspects were the main criteria 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 rather “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 for leading 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 these deserve.

Fig. 1. History of model-based user interface generators.

In the beginning of user interface development, Application Programming Interfaces (API) were chosen for the creation of user interfaces to avoid reinvention and deviation of interaction concepts for each application. More abstraction as well as easier and more flexible definition of interactions became possible by using User Interface Management Systems (UIMS), which interpreted interaction descriptions on runtime.

Better support for design-time modeling was introduced by User Interface Development Environment, some of which are still in use and development.

Targeting direct output of software development artifacts from user- and usage-oriented models through automation, starting in the late 80s, model-based user interface generators have evolved as an answer to usability integration and prototyping. There are many examples for user interface generators that already bring appropriate models with them, e.g. (Janssen et al., 1993).

For such a model-based user interface (UI) development and UI generation, researchers have already suggested approaches 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 et al., 1996). 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 besides their ability for user interface generation.

There are three different types of approaches that have been developed for generating a user interface from a declarative model (Pinheiro da Silva et al., 2000): Interpreters like ITS (Wiecha et al., 1990), UIMS generators like TADEUS (Schlungbaum & Elwert, 1995) or FUSE (Lonzewski & Schreiber, 1996) and source code generators like JANUS (Balzert et al., 1996) and Mastermind (Szekely et al., 1995).

Today, applications are moving from the desktop into the network by Application Service Provision (ASP) and Service-Oriented Architectures (SOA; Erl, 2005). This also has a high impact on user interfaces: when services or whole interactive processes are executed for example using BPEL (OASIS, 2007), BPEL4People (IBM et al., 2007) and WS Human Task (IBM et al., 2007b), user interfaces change from process to process and have to serve as input and output interfaces for complex processes and services assembled from atomic ones. For this reason, generative and model-based concepts are a possible solution, but have to be transferred and enhanced to cope with runtime requirements to form Model-Based Runtime User Interface Environments (RUIE), which can be expected to play a major role in future systems.

In the following, we will explain a concept for advancing towards model-based and generative concepts suitable for user interface generation on design-time, which forms the basis for potential future generative approaches on runtime.

3. An Interaction-modeling Approach

For the analysis and requirements phase, pure object-oriented models like OOA are too far away from the mental models of non-developing stakeholders like users and customers and are too static to describe the processes rather than pure entities in requirements gathering. This is the reason why more narrative and informal representations like use cases have gained importance, which may still be embedded into an object-oriented framework – like UML use cases – but are informal enough to let all stakeholders participate in the creation process.

While UML use cases (Jacobson et al., 1992) are defined as a representation of a systems view (Rumbaugh et al., 1999), task cases – also called task descriptions (Rumbaugh et al., 1999) or essential use cases (Biddle et al., 2001) – have been developed to allow for easier expression and focus more on the user and his/her intentions. Starting with this simple but impressive modeling technique, users and customers can join the analysis process, because they are able to contribute in their own words and with natural language concepts.

This trend shows that flow-oriented, natural language based and semi-structured description methods yield high potential for successful requirements analysis and negotiation to software development projects.

Due to different aspects and people working with task cases and use cases, these two – originally very similar – techniques are often used in parallel, even in modern approaches for software engineering and usability integration, like usage-centered design.

It is also possible to refine, expand and integrate essential use cases into system use cases. Merging both concepts together with the ability of stepwise refinement (top down) or – the other way round – construction from almost atomic interactions (bottom up) provides a sustainable concept for interaction modeling. Thus, development projects are no longer forced to develop two or more sparsely connected models in parallel for interaction flow description.

Fig. 2. Starting with the overall task (top down) or an idea for improvement (bottom up) should both be possible with an incremental model.

Beginning with essential use cases that completely abstract from technical and implementation details (Constantine & Lockwood, 2002), it is possible to identify the task descriptions that are fundamental for the interactive software in development. As they are easy to write and widely recognized by both, user interface designers and software requirements engineers, they form at least a first step in many development approaches.

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 regards 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 and more structured use case model in the complete development cycle, providing a consistent requirement and interaction model and the ability of out-of-the-box user interface generation.

Already task descriptions (Rumbaugh et al., 1999) provide the ability to transform them to a requirement specification or directly use them as initial descriptions for requirements. However, this ability improves much when detailing and expanding them to a complete and more structured use case set. Here, it is important to abandon the idea of writing use cases from a technical or system perspective. Rather the user perspective should be used, when describing the interactions with the system. This makes a parallel description nearly obsolete and provides a strong model, which can be used as a basis in further development steps.

3.1 IML Model Contents

One example for such an extension of the use case paradigm is an XML-based description we call Interaction Modeling Language (IML; Schlegel et al., 2004).

Following different element and structuring proposals of practitioners, which can be found for example in (Ambler, 2000), it is possible to provide an initial structure for a complete use case specification. As we will use it as a basis for a generator approach, we have subdivided

it into three main parts, which are again hierarchically subdivided: Project definition, data definition and use cases.

Figure 3 shows a condensed example view of an IML-Model containing these three parts: Project definition, data definition and only one IML use case.

Fig. 3. Sample IML model (XML-based).

3.1.1 IML Project Definition

The first part provides a project definition that embodies necessary information for handling and the later generation, including actors, roles, stakeholders and natural languages used. Actors and roles are employed for identifying the “user type” interacting with the system in the use case interaction definition. Including stakeholder descriptions in the definition is important to ensure model inherent requirements traceability even if stakeholders leave the development project.

The natural language support allows for easy localization providing a verification or transformation engine with the information necessary for checking model completeness for each destination language. Therefore, all translations of descriptions, notifications etc. in the use case part must be stored together with their interaction context in the model. Translation quality will also benefit directly from the contextual and semantic information available in the model context.

3.1.2 IML Data Definition

Developers often complain about a lack of initial data specification in the field of use cases and similar methods. When this is missing and cannot be added once entities become clearer with the project advancing, interactions are only loosely coupled with the application logic and cannot benefit from the semantics available with the data definition

For this reason we have integrated a facultative data section in IML, which allows specifying implementation-neutral entities like data pools but also predefined types that can be easily transformed into programming language types.

The data definition is initially empty and is completed while performing the incremental use case modeling when data sources and sinks are needed. 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 to define entities that are used in different interaction steps or sequences, e.g. "postal address". For the generation of a complete prototype, it is even possible to define database access at this point.

3.1.3 IML Use Case Definition

The third part is what practitioners normally refer to as use case description, containing 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. These include among others

- unique use case name
- actors / roles interacting
- dependencies
- dynamics
- results
- business rules (e.g. initial credit 1000 Euro)
- additional requirements (e.g. maximum response time)
- review / evaluation marks
- implementation notes

The list should be adapted to the development project type and special process-related requirements, e.g. CMM (Paulk et al., 1993) criteria.

Entries that may be – in some basic form – already included in task descriptions (Rumbaugh et al., 1999) are:

- pre- and post-conditions plus triggers
- purpose / user goals
- frequency of use
- critical scenarios

An additional work context description (Rumbaugh et al., 1999) to identify the context of use (for example "novice at cash desk") is at least partially covered by the actor and role definitions. Preconditions and triggers have to be differentiated: a precondition is a state that has to be active or achieved before being able to start the use case, while a trigger starts the use case actively.

With many facultative parts and wildcard items, IML allows for an incremental development. As the focus for this chapter lies in the interaction description, requirements and consistency elements mentioned above are not discussed in further 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 – as a first step – included and structured in the interaction part of the above IML use case interaction specification, which is mandatory in an IML model.

To achieve this, simple sentences have to be transformed to an XML structure preserving and expanding the information already contained in the standard use case description. If this should be necessary in further project phases, the structured descriptions can easily be re-transformed into natural language statements.

A top level structure, which is already used by use case writers in non-IML use cases is the identification of a use case's usual workflow, which helps to divide the interaction description of a use case into a regular flow and a set of alternative flows (Schlegel et al., 2004) – also called “variants” (Rumbaugh et al., 1999).

Fig. 4. Use case structured with regular and alternative flows.

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

3.2 InteractionCases

So far, the IML concept mainly complies with the standard concept of use cases already used in practice. However, it structures them more consequentially and allows or forces 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 solutions to a problem from the application domain point of view. Also, coherence of interaction and semantic grouping of interactions belonging together is not part of a common use case definition.

Fig. 5. Example for a use case diagram.

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.

Fig. 6. The same Use case diagram with InteractionCases and the InteractionCase flow marked for each actor.

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. On one hand, these actions form a procedure that should not be split up further in order to concentrate all information necessary for an interaction, meaning to make the InteractionCase as big as necessary. On the other hand, an InteractionCase shall not contain steps that belong to a different context, making it as small as possible. This concept transfers the principle of strong internal cohesion and loose coupling with other components from Information Hiding (Parnas, 1972) 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 the figures, the steps necessary to enter a complete address together 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 like 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 additionally can be grouped semantically within their InteractionCase.

Fig. 7. Sample structure of an IML use case definition.

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 window 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.

3.3 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 figures 8 to 12). These elements can be used directly or be recursively grouped by semantic group elements structuring the interaction flow. Three of the basic elements are described in the following.

Fig. 8. Structure of use cases and InteractionCases.

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.

Fig. 9. Concept of the ENTER action.

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.

Fig. 10. Concept of the EDIT action.

SELECT actions let the user choose 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.

Fig. 11. Concept of the SELECT action.

WRITE is used to describe – mainly textual – system output. Like for the input actions, the sort of output is specified by the type of the content, for example “natural number, range 1 to 31”. To be implementation neutral, the selection of display options is accomplished in the TARGET description.

Fig. 12. Concept of the WRITE action.

TARGETs are a sort of interaction entity: All interactions implicitly or explicitly specify their target interface over which the user interacts with the system. Speaking in terms of a graphical user interface, one target could represent a status bar. If no target is indicated, each interaction uses the standard target that is defined with each InteractionCase.

GENERIC Actions: As long as the type of an action is not clarified – like when switching from textual to IML description – it is described with a GENERIC action, which can be easily transformed into or overwritten by any concrete action.

3.3.1 Sub-Structuring with IML Groups

Often a further sub-structuring mechanism is needed to create blocks inside an InteractionCase, which emphasizes the higher coherence of e.g. first name and last name compared to first name and street in an address. The semantic group element allows for a hierarchical structuring of elements as well as groups and provides description mechanisms that help the user interface designer or generator to 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 per InteractionCase is facilitated and helps making suggestions for dialog structure improvement on a semantic basis already before dialogs are finally created.

3.3.2 Textual Descriptions

Besides the texts used for direct interaction, each interaction element in IML has attached textual descriptions. Specifying an actor or stakeholder for which the information is intended, like for example designer, developer or user and a representation context like *integrated*, *online-help*, *printed* etc. it is possible to differentiate between different target artifacts. This will play an important role in the artifact generation later on. Anytime an online-help or developer documentation is needed, it can be extracted from the model. A profile for each user group will specify which texts (starting from which priority) will be extracted or used for extraction and which language they have to be in.

One of the concept's strengths is the verification capability that comes with such an integrated description: the completion criterion per language. For example, setting the user-related descriptions criterion for "German" to "complete", a tool will be easily capable of checking all descriptions of this type for containing a description in the desired language with the desired state, e.g. "evaluated". Employing this mechanism will lead to fewer problems and lacks in translation to new spoken languages and enforces a complete model-contained documentation.

3.3.3 Actors

InteractionCases usually address only interactions with some of the actors in the complete set of actors referring to the use case, whereas every interaction may again affect only some of these actors. For a consistent model, it is important, that all actors used in an entity are defined on the above level in the model:

actors $\supseteq$ actors(use case) $\supseteq$ actors(InteractionCase) $\supseteq$ actors(interaction)

3.3.4 System Actions

While EDIT, ENTER and SELECT focus on the user perspective of the interaction sequence, PERFORM represents the system actions. This “double-perspective” modeling is necessary to integrate the user and the system view in one model. Though, these actions cannot be viewed mutually independently: A user action has a coupled request on the systems side, while many system actions will be perceived by the user through the user interface. The next section describes how these concepts can be used for automation.

4. User Interface Generation

Since the early nineties there has been a struggle for developing models and generators for user interface generation. Although some of them have been very promising, none of them has been widely used in practice.

Why did no strong use in practice happen in the last decade? One major answer to this question lies in the fact that often completely new models have been introduced that were not known in practice so far. Analysts and developers would have been forced to learn new modeling techniques and paradigms before seeing a benefit of their efforts. This makes user interface generation a good example for highly developed and specialized approaches that have not found their way into practical software development due to strong requirements, new partial models and missing integration with other aspects of software engineering and user interface design. Although, the results of user interface generator development have had and still have high impact on the research done and might even increase in the dawn of SOA.

The IML approach presented tries to avoid using a completely new model. Textual descriptions of standard use cases can be transferred to IML use cases by creating at least one InteractionCase per use case and transforming every single item or line of the textual description into one generic IML action. Assignment of types, grouping and partition into more InteractionCases can be carried out in a refinement step later on. With the integrative modeling concepts of UML 2 (OMG, 2007; OMG, 2007b) integration into the software engineering landscape will become even easier.

4.1 Prototype Generation from IML Models

The problem of a pure model-based approach and the creation of a prototype based on this model is the long time elapsing between the start of modeling and first user or customer feedback. Our goal was therefore to preserve the advantages of a model-based approach while minimizing the “time to prototype”, which leads to the goal of a rapid and integrative model-based user interface prototyping.

IML provides abilities for transformation into an abstract user interface definition directly from an IML-enhanced use case model.

Once a model is completed for a first iteration, concrete interaction elements for abstract definitions must be found (Constantine et al., 2003). To have influence on the final representation, every automated generation process should create an intermediate model or abstract user interface definition, which can be used as basis for decisions or further generation steps, which are necessary in design time UI generation.

A transformer for IML or similar models should work with a transformation description that returns the realizing element type or at least a set of possible realizing element types for

a specified interaction and context. The element type is then attached to every abstract component. This makes the generation of the final code the main challenge after the element type has been selected.

Fig. 13. Transforming specification into a UI prototype.

Layout generation can only be accomplished when element geometry has been selected or is already provided with the transformation description. An overview of layout criteria and rules can be found for example in (Hofmann, 1998).

Stepping over to an object-oriented software design is an alternative to direct prototype generation, especially for large projects and where generation is not available or not applicable.

Using IML it is possible to generate or derive the navigation structure from the model. For example, menus can be synthesized from some use cases that are referenced by the system or basic use case. The interaction flow in a wizard or another structured interaction sequence can also be compiled directly from the interaction specification in an IML model.

4.2 Definition of an Abstract User Interface

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). DiLL is specialized on graphical user interfaces, meaning that for other interaction modes like Voice User Interfaces (VUI) a different and specialized intermediate model will be needed. The Dialog Layer Language is intended for graphical and textual user interfaces only, because it deals with screens, groups and elements related to parallel viewing.

Such an intermediate modeling language is necessary to allow for manual changes in the generated user interface definition without manipulation of code fragments and resource definitions. For such an abstract user interface definition, it is necessary to have a very basic structure with few elements.

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. It contains only SCREENs, GROUPs and ELEMENTs, which all bring types and geometry with them.

Fig. 14. DiLL SCREEN, GROUPs and ELEMENTs in a sample dialog.

SCREENS provide a sort of empty board like a window or screen, which can be filled with elements. ELEMENTs are parameterized by element type, coordinates etc. Visible and semantic grouping is achieved by the use of the GROUP element.

Fig. 15. Transformation Process.

4.3 Prototype Generation Process

The user interface generation process presented here follows a pipeline approach to allow linear transformation and to accomplish the different tasks from model completion to code generation. It is very similar to the visualization pipeline in computer visualization. A transformer prototype for this process has been created covering the following steps:

Completion: Usually, an IML model is not fully completed at generation time due to incremental development. Or it needs completion steps – like generating 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.

Transformation: The transformation step interprets the IML model and forms a DiLL model based on 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: now a generic user interface has been created on the top of the IML specification.

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) of elements on one aggregation layer (like group or window) to complex calculations and conditions depending on field types and their translation to composite elements.

Generation: The optimized DiLL model is the ideal basis for the code generation. For each generation target system a unique Element Transformation Description (ETD) file exists, which describes 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 Element Transformation Description (ETD) format is used to describe code insertion into a specific template for a programming project like those of MS Visual Studio. It 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.

5. Generation of other Final Artifacts

So far, we have only discussed the ability to generate the final artifacts¹ user interface prototype, help and documentation from IML models. But a fully staffed IML model provides enough information to allow for generation of artifacts beyond GUI models.

¹ We define final artifacts as all artifacts that experience no further transformation and are therefore in their final state – unlike source models or intermediate representations.

Of course, a straightforward sequence of transformations from model to model can be achieved more easily and complies with the requirements of a waterfall model or a stepwise usage-centered design (Constantine et al., 2003). However, necessary iterations occur in most software development projects because of the ambiguous requirements set: At the beginning of a project, the current state and some major requirements and changes are clear. But a complete set of requirements, which can be used throughout the whole project, is not the usual case. Therefore, model amendments and cycles are nearly inevitable. Supporting iterative development is therefore one major requirement for integrated models.

Methodic and well-founded model creation is crucial for project success. But as human beings make mistakes and cannot write down a complete requirement set at once, the key is to avoid frequent model changes in all steps but to create multiple scenarios in the same cycle and again for the next iteration. Once a model has been agreed upon, a prototype can be generated using a user interface prototype generator. This makes the real iteration very efficient, because a mistake in the model does not lead to cascading changes.

7. Integrated Engineering of User-oriented and Traditional Models

Many problems regarding the integration of user-oriented models and software engineering models result from a stepwise concretization: Implementation neutral models are transformed into specific models making design and implementation decisions. Often, models of different abstraction layers turn out to be incompatible. For this purpose, different methods for manually transferring an initial use case definition to an object-oriented design have already been described (Biddle et al., 2001). To ensure traceability and interconnections between task and object models, many approaches use responsibility as the driving factor.

7.1 Model Integration

Fig. 17. Interconnection of different partial models in one holistic model.

An alternative is a model integration that interconnects artifacts of different partial models and abstraction layers directly using different forms of associations. In this way, collaborations and responsibilities can be modeled in an integrative way with direct references to the emerging classes created to implement them.

Unfortunately, the essential use cases are often only loosely interconnected with other models that are developed in parallel – e.g. role models – and also in no way have a connection to models of subsequent phases or realization layers.

An InteractionCase or at least a complete Use Case should for example have a link to the implementing dialog classes in the design description, e.g. in the class diagram.

7.2 Generative Models (Deduction)

Another 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 users and customers. Originally, we have used this approach for the user interface generator. It turned out to be applicable when a pure prototyping was done, which allowed us to overwrite the intermediate model and the code prototype with each generation.

Used in iterative and agile development approaches, often changes have already been made in these artifacts. This is where partial code-preserving generation is needed. The effort necessary to join new and old code makes generation inapplicable. A partial generation using an integrated model could solve some of these problems.

7.3 Concurrent Engineering

Concurrent Engineering for usage-centered design processes (Constantine et al., 2003) should be limited to models that have only rare dependencies. While in the usage-centered design process (as proposed in: Constantine et al., 2003) roles and system actors are developed in parallel with only a conjunction over a domain model, a more integrated approach should create a role model that is used in the use case model together with the actors (actors performing one or more roles). It is necessary to distinguish system actors (i.e. systems that play an actor role) from user actors. But as it may not always be clear which interaction comes from systems and which from users, this should not lead to a completely different use case description, if not necessary. As long as users are involved, the description should be quite similar to make them understand the scenario.

Indirect actors that only have impact on the system mediated by other (direct) actors should exist in the role model but in no case be part of a modeled system interaction to avoid confusion in the model.

As an actor may play different roles and sometimes one role may be played by different actors, the current actor's role should always be provided with the USES link or ACTOR in an integrated model. Alternatively, it is possible to create a new actor entity for each actor/role combination.

It is often criticized that user centered designers tend to go too quickly into paper prototyping, whereas in usage-centered design a content model is developed that covers the overall organization of the user interface. The advantage of an integrated model-based

approach in this case is obvious: Mistakes and inconsistencies made in the overall interface architecture can be found using prototypes generated from the model. On the other hand, these findings can again have a direct impact on the model, which unites the advantages of model-based and user centered approaches.

As we can learn from the software engineering process discussion, pure straightforward processes like waterfall can only be used in very specific cases in software development practice. Therefore, one pitfall of straightforward user interface generation approaches often has been the missing back link of the artifacts generated in the process.

This lack of model interconnectivity does not allow corrections in intermediate and final artifacts, because impacts are not visible and a second generation cannot cope with changes made or even overwrites them.

A simple but powerful solution to this problem can be achieved through generating artifacts in a holistic model that contains source model and target artifacts. For a model-internal transformation process it is easy to construct links from a specification artifact to the generated prototype artifact, additionally referencing the transformation used.

Especially for web interfaces such a representation is useful, because it provides very fine-grained and component-based artifacts, which form a complete page when synthesized by a simple export routine. The result of a generation with rules can be any kind of text-based document including ASCII text, source code, HTML or any kind of XML. Once the structure and artifact components are defined, the process is the same.

8. Soft Artifact Engineering

The IML and similar approaches can be extended towards a holistic engineering approach for soft artifacts, which integrates processes, stakeholder perspectives and different model abstraction layers of software and service engineering projects into one model. Such a component- and flow-oriented approach joins the more static perspective of e.g. inheritance and aggregation with the dynamics of interaction, process and artifact flow. Object-oriented paradigms allow for the use of inheritance/classification and template concepts for every artifact type like one of the InteractionCases presented.

To allow for this integration, we propose a slightly different view upon the different models, which we call partial ones. Using one meta-model like MOF (OMG, 2002) for defining each model and one development environment supporting the meta-model and the defined partial models, it is possible to interconnect all partial models, though having all possibilities of the stand-alone models.

This view can be used for all virtual – i.e. “soft” – artifacts, like use cases, dialog designs, software components and services. For this reason and for its ability to provide an approach integrating model and process, we call this approach Soft Artifact Engineering (SAE).

SAE is a holistic and integrative way to allow for 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 advantage of these efforts on integration and will lead to an integrated interactive system development approach with abilities for intermediate models and internal user interface prototype generation.

The expansion to other domains like Service Engineering will allow developing services, user interfaces and software in one closely interconnected common model.

9. Summary and Outlook

In this chapter we have provided an overview on how user interface modeling and generation can be approached and how they can be integrated with software engineering. For this purpose, we have presented an approach for better integration with the software engineering models. This is achieved by a process-oriented modeling approach based on (UML) Use Cases. The resulting models have then been used in a pipeline generation process, which supports iterative and integrated development of interactive applications. The process and prototype tool developed offer automated model completion, generation of an intermediate dialog layer language and generation of final artifacts – such as a Visual Studio project containing the interface code and resources as well as help and documentation. The chapter explains the concepts of the model and the generation process, to allow for transfer and application to other model-based user interface generation approaches.

Current developments in software engineering and business IT show a trend towards regarding applications not as fixed and local installations. Instead, a process towards virtualization and decentralization takes place for computing and data as well as business processes and especially services – often referred to as “cloud computing”. With dynamically changing services, processes and compositions in decentralized runtime architectures, design-time modeling and generation will not be sufficient anymore.

In the frame of INT-MANUS project (Schlegel & Thiel, 2007) we have been able to prove that processes in production can be executed in a fully decentralized system, generating actions and interactions for each process instance just-in-time locally (Schlegel, 2008). To be able to generate interactions, dedicated semantic models will be needed to derive context and requirements from a classification and interaction model.

First approaches like BPEL4People (IBM et al., 2007) and WS Human Task (IBM et al., 2007b) show the path towards interactive processes and applications deployed and running dynamically in future networked systems. Applying semantic model-based concepts to these environments will offer new possibilities for runtime composition and generation of user interfaces for services in such a cloud and for dynamic and distributed applications in general.

Acknowledgements

The author wishes to thank Dr. Alexander Burst from ETAS GmbH and his former colleagues from the Fraunhofer Institute for Industrial Engineering (IAO) for their kind support of this work. Application and integration of this research has been done in the projects LIKE and KOMPASS funded by the German Ministry of Education and Research (BMBF) as well as INT-MANUS project and I*PROMS Network of Excellence funded by the European Commission under the Sixth Framework Programme.

10. References

- Alexander, I. & Robertson, S. (2004). *Understanding Project Sociology by Modeling Stakeholders*, IEEE Software, 21(1), January 2004, pp. 23-27
- Ambler, S.W. (2000). *Documenting a use case: What to include, and why*, <http://www-106.ibm.com/developerworks/webservices/library/ws-tip-docusecase.html>, 2000
- Balzert, H.; Hofmann, F.; Kruschinski, V. & Niemann, C. (1996). *The JANUS Application Development Environment – Generating More than the User Interface*, In: *Computer-Aided Design of User Interfaces*, pp. 183-206, Namur University Press
- Barclay, P.; Griffiths, T.; McKirdy, J.; Paton, N.W.; Cooper, R. & Kennedy, J. (1999). *The Teallach Tool: Using Models for Flexible User Interface Design*. Proceedings of Computer-Aided Design of User Interfaces II (CADUI'99), pp. 139-158, Kluwer
- Beck, K. (1999). *Extreme Programming Explained*, Addison-Wesley
- Biddle, R.; Noble, J. & Tempero, E. (2001). *Essential Use Cases and Responsibility in Object-Oriented Development*, Technical Report, CS-TR-01/01
- Coad, P. & Yourdon, E. (1991). *Object-Oriented Analysis 2nd Edition*, Yourdon Press, Englewood Cliffs N.J.: Prentice-Hall
- Constantine, L.L. & Lockwood, L.A.D. (2002). *Usage-Centered Engineering for Web Applications*, IEEE Software, 19(2), pp. 42-50
- Constantine, L.L. (1995). *Essential modeling: use cases for user interfaces*. ACM interactions, Vol. 2, Issue 2, pp. 34-46, New York: ACM Press
- Constantine, L.L.; Biddle, R. & Nobel, J. (2003). *Usage-Centered Design and Software Engineering: Models for Integration*, Proceedings of ICSE 2003, pp. 106-113
- DeMarco, T. (1979). *Structured Analysis and System Specification*, N.J.: Prentice-Hall, Englewood Cliffs
- Erl, T. (2005). *Service-Oriented Architecture – Concepts, Technology, and Design*. Prentice Hall
- Firesmith, D.; Henderson-Sellers, B. & Graham, I. (1998). *OPEN Modeling Language (OML) Reference Manual.*, New York: Cambridge University Press, Cambridge
- Harel, D. (1987). *Statecharts a visual formalism for complex systems*. In: *Science of Computer Programming 1987*, pp. 231-274, Elsevier Science Publishers
- Henderson-Sellers, B. & Firesmith, D.G. (1999). *Comparing OPEN and UML: the Two Third-generation OO Development Approaches*. Information and Software Technology, Vol. 41, No. 3, pp. 139-156
- Hofmann, F. (1998). *Grafische Benutzungsoberflächen: Generierung aus OOA-Modellen*. Spektrum, Akademischer Verlag, Heidelberg, Berlin
- IBM et al. (2007). *WS-BPEL Extension for People (BPEL4People)*, Version 1.0, June 2007
- IBM et al. (2007b). *Web Services Human Task (WS-HumanTask)*, Version 1.0, June 2007
- ISO (1999). *Human-centred Design Processes for Interactive Systems (ISO standard 13407: 1999)*
- Jacobson, I.; Christerson, M.; Jonsson, P. & Övergaard, G. (1992). *Object-Oriented Software Engineering: A Use-Case Driven Approach*, Addison-Wesley
- Jacobson, I.; Christerson, M.; Jonsson, P. & Övergaard, G. (1992). *Object-Oriented Software Engineering*, Addison-Wesley, Reading, Mass
- Janssen, C.; Weisbecker, A. & Ziegler, J. (1993). *Generating User Interfaces from Data Models and Dialogue Net Specifications*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 418-423, April 24-29, Amsterdam, The Netherlands

- Janssen, C.; Weisbecker, A. & Ziegler, J. (1993). *Generating User Interfaces from Data Models and Dialogue Net Specifications*, Proceedings of INTERCHI'93, pp. 418-423, ACM Press
- Larson, J.A. (1992). *Interactive Software – Tools for Building Interactive User Interfaces*, Yourdon Press Computing Series, Prentice-Hall
- Lauesen, S. (2003). *Task Descriptions as Functional Requirements*, IEEE Software, 20(2), pp. 58-65
- Lonczewski, F. & Schreiber, S. (1996). *The FUSE-System: an Integrated User Interface Design Environment*, Proceedings CADUI'96, Namur University Press
- Martin, J. & Odell, J. (1992). *Object-Oriented Analysis and Design*, N.J.: Prentice-Hall, Englewood Cliffs
- Metzker, E. & Seffah, A. (2003). *User-Centered Design in the Software Engineering Lifecycle: Organizational, Cultural and Educational Obstacles to a Successful Integration*, Proceedings of HCI International 2003, Human-Computer Interaction, Theory and Practice (Part I), pp. 173-177
- 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
- OASIS (2007). *Web Services Business Process Execution Language Version 2.0*, OASIS Standard, 11 April 2007, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>
- OMG (2002). *Meta Object Facility (MOF) Specification*, OMG-Meta Object Facility, v1.4, <http://www.omg.org/cgi-bin/doc?formal/2002-04-03>
- OMG (2007). *Unified Modeling Language (UML) Infrastructure Version 2.1.1. formal/07-02-06*, 2007, <http://www.omg.org/cgi-bin/apps/doc?formal/07-02-06.pdf>, 28.3.2007
- OMG (2007). *Unified Modeling Language (UML) Superstructure Version 2.1.1. formal/07-02-05*, 2007, <http://www.omg.org/cgi-bin/apps/doc?formal/07-02-05.pdf>, 28.3.2007
- OMG. (2003). *UML 2.0 Superstructure Specification*, OMG Final Adopted Specification, <http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.pdf>
- Parnas, D.L. (1972). *On the Criteria To Be Used in Decomposing Systems into Modules*. In: Communications of the ACM, Vol. 15, No. 12, December 1972, pp. 1053-1058
- Paulk, M.C.; Curtis, B.; Chrissis, M.B. & Weber, C.V. (1993). *Capability Maturity Model, Version 1.1*, IEEE Software, 10(4), pp. 18-27
- Pinheiro da Silva, P. (2000). *UMLi: Integrating User Interface and Application Design*. UML2000 Workshop on Towards a UML Profile for Interactive Systems Development (TUPIS2000), York, United Kingdom
- Pinheiro da Silva, P.; Griffiths, T. & Paton, N.W. (2000). *Generating User Interface Code in a Model Based User Interface Development Environment*, Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI2000), pp. 155-160, ACM Press
- Pressman, R.S. (1997). *Software Engineering – a Practitioners's Approach, European Adaptation*, Fourth Edition. New York: McGraw Hill
- Reisig, W. (1986). *Petrinetze – Eine Einführung*, Berlin, Heidelberg, Springer-Verlag
- Rubin, K.S. & Goldberg, A. (1992). *Object behaviour analysis*. Communications of the ACM, Vol. 35, No. 9, pp. 48-62
- Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F. & Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Englewood Cliffs N.J.: Prentice-Hall
- Rumbaugh, J.; Jacobson, I. & Booch, G. (1999). *The Unified Modeling Language Reference Manual*, Addison-Wesley

- Scheer, A.W. (1994). *Business Process Engineering. Reference Models for Industrial Enterprises*, Berlin: Springer
- Schlegel, T. & Thiel, S. (2007). *Semantic Models and Processes for Information Gathering and Interactive Workflow Execution in Decentralized Production Environments*. In: *Intelligent Production Machines and Systems - 3rd I*PROMS Virtual International Conference*, 2007
- Schlegel, T. (2008). *Interactive Processes and Semantic Models in Production Systems*, Proceedings of the First International Conference on Business Sustainability (BS08), Ofir, Portugal
- Schlegel, T.; Burst, A. & Ertl, T. (2004). *A Flow Centric Interaction Model for Requirements Specification and User Interface Generation*, Proceedings of the 7th International Conference on WWCS, Bridging Diversity at Work, Damai Sciences, Kuala Lumpur
- Schlunbaum, E. & Elwert, T. (1995). *Modellierung von Graphischen Benutzungsoberflächen im Rahmen des TADEUS-Ansatzes*, In: *Software-Ergonomie '95*, Böcker, H.-D., pp. 331-348, Teubner, Stuttgart
- Scott, K. (2004). *Fast Track UML 2.0*, Apress
- Szekely, P.; Sukaviriya, P.; Castells, P.; Muthukumarasamy, J. & Salcher, E. (1995). *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*, Proceedings of the EHCI'95, pp. 120-150, Chapman & Hall, London
- Vredenburg, K.; Isensee, S. & Righi, C. (2002). *User-Centered Design: An Integrated Approach*, Prentice Hall
- Wiecha, C.; Bennet, W.; Boies, S.; Gould, J. & Green, S. (1990). *ITS: A Tool for Rapidly Developing Interactive Applications*. ACM Transactions on Information Systems, Vol. 8, No. 3, pp. 204-236
- World Wide Web Consortium, W3C (2004). *Resource Description Framework (RDF)*, <http://www.w3.org/RDF/>

Human-Computer Interaction

Edited by Inaki Maurtua

ISBN 978-953-307-022-3

Hard cover, 560 pages

Publisher InTech

Published online 01, December, 2009

Published in print edition December, 2009

In this book the reader will find a collection of 31 papers presenting different facets of Human Computer Interaction, the result of research projects and experiments as well as new approaches to design user interfaces. The book is organized according to the following main topics in a sequential order: new interaction paradigms, multimodality, usability studies on several interaction mechanisms, human factors, universal design and development methodologies and tools.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Thomas Schlegel (2009). Model-based User Interface Generation from Process-Oriented Models, Human-Computer Interaction, Inaki Maurtua (Ed.), ISBN: 978-953-307-022-3, InTech, Available from: <http://www.intechopen.com/books/human-computer-interaction/model-based-user-interface-generation-from-process-oriented-models>

INTech

open science | open minds

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2009 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](https://creativecommons.org/licenses/by-nc-sa/3.0/), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.