ABSTRACT

Automated generation of application documentation targeted to the end user is sparsely investigated. This is partly due to the fact that, up to now, no application model existed, providing the required comprehensive information about the application. The new generation of Eclipse based Rich Client Platform applications is developed on the basis of an application model, which for the first time closes this gap.

Due to this advance we investigate on the possibilities and limitations on employing the application model for automated documentation generation. This paper outlines the current position on this topic, the research questions arising and both the scientific and technical approach to tackle this question.

This research is specific to the Eclipse platform only, as other technologies currently do not employ the required comprehensive application model. The Eclipse application model, however, can also be used for non-Eclipse based applications, such that the results found are transiently applicable to other solutions.

KEY WORDS
Software Design and Development, Software Design Tools, Authoring Tool, Application Documentation

1 Introduction

While the automation of developer targeted documentation is often embedded into programming languages (e.g. Java: Javadoc or Python: Pydoc), and there exist tools to help the programmer in using those (e.g. Java: JAutodoc or Python: Epydoc), the same aspects are barely studied much less available for application software\(^1\) documentation targeted at the end user. This is partly due to the fact that the meaning, respectively the semantics of an application cannot be automatically derived from its source code and the required information to auto-generate an application documentation hence simply is not available.

Applications based upon models perform better regarding this requirement, as the model provides a more abstract description of an application’s meaning and component inter-connections. However, models up-to-now are mostly used to manage specific aspects or parts of an application, but seldom to create the full application itself.

The new Eclipse Rich Client Platform generation (Eclipse 4) is entirely based on the Eclipse Modelling Framework (EMF). This tight integration has led to a comprehensive application model providing information about the structure and inter-connections of the application’s components, thus filling a significant gap that hindered further advances on automated application software documentation generation.

This paper outlines the research path on determining to what extent the Eclipse application model can be used for automated application documentation generation.

2 State of the Art

Documentation of an application is targeted at different audiences and different layers of the application. Developer focused documentation, called software documentation, for example includes source code and design decision documentation. Documentation focused on the usage of an application, application documentation, is targeted at the end user.

2.1 Documentation Generation

The basic concept of combining documentation and code, coined literate programming, dates back to 1984 [1] and has been treated in several research projects and papers already. Literate programming generates static software documentation out of the source classes and packages. Being reduced to source classes and groups of classes (packages) only, severely limits the explanatory power about the full application and is thus not applicable to application documentation.

Johnson et al. [2] propose a development documentation, on-line assistance system for software maintainers and other software professionals, called I-Doc. This system dynamically generates documentation in response to specific requests for user information. It is focused on supporting the re-engineering of software systems. I-Doc covers the

---

\(^1\)ISO/IEC 26514 defines application software as software designed to help users perform particular tasks or handle particular types of problems, as distinct from software that controls the computer itself.
comprehensive codebase of the application, providing support for the engineers software understanding activities. It is thus a generator for a comprehensive software documentation.

Work on automated application documentation generation can be found in [3] and [4].

In [3] a model forms the base for documentation generation. ISOLDE (Integrated Software and On-Line Documentation), an automated hypertext based on-line help starting from software specifications, generates text from task models generated by a Computer Aided Software Engineering tool (here RationalRose). Project ISOLDE ended in 2002. ISOLDE covers instructional mode, task oriented documentation (cf. figure 1) representing a small part of application documentation only. The model, the documentation is generated from, is maintained separately from the applications source code.

The authors of [4] use a dynamic analysis method to generate end user documentation out of the product (in this case a Web 2.0 application). The outcome of their method is an IEEE 1063 conform application documentation. The main problem treated in [4], is the retrieval of the necessary comprehensive application information. The authors employ finite state machines to generate navigation graphs out of a users behaviour during the usage of the application. The sum of these navigation graphs is then semi-automatically processed to create the application documentation. This problem described is solved in our scenario through the application model, readily providing the comprehensive application information.

### 2.2 Documentation specification

ISO/IEC 26514 [5] defines requirements for the design and development of application software documentation as part of an application’s life cycle process. It defines this process from the viewpoint of the documentation developer and also covers the documentation product with its structure and characteristics. Figure 1 gives an overview of these characteristics.

![Figure 1. Mode characteristics of application software documentation](image)

The approaches as documented in section 2.1 are each only applicable to a subset of those characteristics, and hence are not able to satisfy the requirements of a comprehensive application software documentation as documented in ISO/IEC 26514 [5].

Besides ISO/IEC 26514 [5] there also exist non-formal documentation guides like [6] which are mostly helpers recommending a certain writing style and providing specific examples. The formal content in [6] is for example derived from IEEE 1063 [7] (as used by the authors of [4]) which has been superseded by ISO/IEC 26514 [5].

### 3 Materials and Methods

#### 3.1 Eclipse application model

The Eclipse environment forms the base for several scientific and business projects. One of its most famous outcomes is the Integrated Development Environment (IDE) [8]. Eclipse Rich Client Platform (RCP) is a platform for building and deploying rich client applications. It was extracted as a self-contained project from the original Eclipse IDE project in 2003 and serves now as the base for the IDE itself.

Eclipse 4 superseded the seasoned Eclipse 3 framework as new stable in 2012 with its Juno release. Eclipse 4 is based upon the EMF [9] core and hence being developed based on means of model technology. Fundamental to this is the Eclipse application model, denoting the components and the structure of the application itself. This application model (an example is presented in figure 2), contains different types of entities which can be classified into acting entities (AE), visual presentation entities (VPE) and application construction entities (ACE).

AEs perform a certain task that changes the state of the application (e.g. the abstract command “add a person”, as shown in fig. 2, changes the application state by adding a new person object).

VPEs denote how the application is presented to the user (e.g. the viewpart “person list”, as shown in fig. 2 shows the list of persons known to the system). VPEs also contribute AEs in the form of entities the user may interact with (e.g. buttons, menu items, etc.).

ACEs convey information relevant to the internal workings that are not directly presented to the user (e.g. the “command categories”, as shown in fig. 2, could be used to group all commands related to modifying the list of persons into a “person management” group).

Figure 3 provides an overview of the interconnection between VPEs and AE in the application model. Application is the parent element of the application model (cf. fig. 2) containing VPEs of type Part (which is a View) and Perspective (which structure parts). Each AE is contributed in some way by a VPE, as it is otherwise not callable from within the application. The Handler implements the Command and thus refers to the actual binary class executing the action. There can be more Handlers associated to a Command, but only one is active given a specific application context.

ACEs are meta-entities with respect to this figure and hence not depicted.
3.2 Development process

Each application model entity owes its existence to a certain business requirement, user story or use-case to be implemented by the developer during the development process. According to [5] Development of the user documentation should be part of the same processes as the software product life-cycle, and ideally performed in conjunction with development of the software.

The Eclipse application model, as described, is generic and not limited to the Eclipse technology base. There already exists a so called renderer framework [10][pages 369 - 375], that allows the creation of Eclipse application model based applications rooted in different technologies.

4 The Research Questions

The overall question posed in this project is What are the possibilities and limitations of automated ISO/IEC 26514 conform application documentation generation based upon the Eclipse application model.

This question can be broken down into the following sub-questions arising by the target of folding application model and application documentation into a common development base:

4.1 Combining documentation artifacts and application model

Figure 4 gives an overview on the processes leading to an application, its documentation and the generated artifacts.

The Agile Development process creates, among other artifacts, user-stories which identify the audience of the application and the tasks the application has to be able to execute, as well as their meaning. Implementation leads to elements in the application model such as AEs and VPEs backed by their resp. implementation classes.

A user story thus leads to a task executed by a specific audience (user). This task is executed as a procedure in a series of ordered steps, where each step is an action. An action is executed by an AE located within the application model, and the resp. action trigger (in this case a button) is visible to the user by means of a VPE.

As there exists a 1-to-1 relationship between an AE in the application model and an action within the documentation we have an initial formal connection point. So the resp. AE could be enriched by its semantic description, which is the natural language description of the action of an AE or the content presented by a VPE.

The specific research question here is, to what extent the connection of elements in the application model and application document is extendable. For example: An acting entity can be part of a group of (unordered) acting entities.
in the application model (namely Command and Command Category, cf. figure 2), can this connection be extended such that the resp. entity group may represent a Procedure?

4.2 Mapping the application model to the documentation artifacts

The research question posed in section 4.1 takes a very basic approach to identify connections between application model and documentation artifacts, at which stage the concrete structure and content of the application documentation does not yet come into play.

[5] defines the process in which the artifacts identified for the documentation are processed to form the resulting documentation.

Figure 5 provides a schematic description depicting this process. It starts with the collection of documentation artifacts such as Audience, Tasks, Error-Messages, etc., and the subsequent processing of this information.

The main artifacts generated during this process are the Audience Mapping Matrix with a series of Information profiles and a number of ordered task lists, weighted according to different criteria such as task difficulty or frequency of occurrence.

As at the development stage, AE and VPE become instantiated into their respective application model implementations, representing an action within a task. They are enriched with their semantic description which provides the connection to the documentation artifacts.

The main research question posed at this stage is, whether the application model currently given is expressive enough to allow for the mapping of documentation artifacts. That is, is an extension of the existing application model element necessary, and is it required to introduce new application model elements? A specific example to this is the Error-Messages artifact in the documentation. It is connected to a task (or action) and denotes a possible erroneous application state with a defined error number and a series of steps for the solution of this problem. In Java, the existing programmatic representation of this is an Exception. So if, for example, a new application model element representing an ApplicationException would be added, would it suffice to contain the required mapping between application model and documentation artifacts?

4.3 Combining development and documentation

Documentation and Application are both constant subject to change and hence need to be tightly connected. An earlier version of an application normally features a reduced set of functionality with respect to a later version of the documentation.

Due to that, a documentation created for an earlier version is only of limited value for an up-to-date application. As the target is to combine application model and documentation into a common model, the main problem will be the differential part of the documentation not directly allocated to a model element. An example to this is the introduction chapter, required by the documentation, explaining the general usage of the application. In an Eclipse RCP application this would be the overall structure as provided by the framework (where we have concepts like Views and Perspectives) which may change between its revision. So there has to be some base-documentation part delivered by the framework in advance to cover for this.

There is also the problem of product modularity. That
is, the assembly and functionality of an Eclipse RCP application strongly depends on the modules it is delivered with.

An initial viewpoint is to see the product itself as non-changing in functionality in between versions. It has to be considered, however, that the documentation is to be generated out of the features sets, not the comprehensive product.

Figure 6 gives an overview on the elements an Eclipse application is created from. An Eclipse product may be based upon features or bundles directly. As bundles themselves do not support the Eclipse provisioning platform (p2) [12][pages 335-351], they are generally not directly (that is without allocation within a feature) used in software distribution.

Considering these problems we subsume the research question in this part to: How to combine the Eclipse RCP development with the documentation development, considering the dynamic structure of a delivered product.

4.4 Determine the solutions applicability

As the system is targeted at populating the documentation due to the elements provided in the application model, it is clear that the less information is available in the model, the sparse the documentation will be.

It is not the target of the solution to relocate programmatic parts into the application model, as to make them accessible for the documentation, but to provide a framework which leaves only the “last-mile” of the documentation for hand-writing.

This is in accordance with tools like EMF which provide a lot of prepared boilerplate code out of the data model, but are far from “automatic application creation”.

It is an open question to what types of applications a sufficient result may be achieved in terms of a usable documentation. The materials used, however, limit the targeted application types somewhat, as can be seen in figure 7.

Here we allocate applications depending on their manufacturing method and the distance to the hardware. An operating system device driver, for example, is clearly not targeted to the documentation system (and does not use the materials we base the solution on).

So the focus lies in the area of business solution applications, with an unclear boundary to system-near individual software and standard application software.
Figure 7. Types of applications with resp. to manufacturing method and distance to hardware.

It is necessary to identify representatives of these application types as to have a meaningful base to benchmark the solution. This also accounts for different application types within the business solution application area (e.g. is the solution as applicable for an electronic medical record software, as it is for a mindmap tool?).

5 Technical Approach

In order to develop the automated documentation generation, we fix a sample application scenario to support the processes shown in fig. 4 and 5. This sample application is settled in the medical domain, considering a (reduced) pharmacy following local regulations.

There are 4 audiences in this scenario (seller, stockist, clerk, administrator) with an average of 6 tasks modelling a vending scenario. By assuming this sample application, we generate the full set of artifacts and elaborate on their usage in the automated documentation generation, where the initial extension to the Eclipse development environment will be applied as follows:

5.1 Develop the required tooling

The first step is to fork the Eclipse application model tooling in order to add support for the concept of semantic description to the application model elements. Subsequently the connection between a documentation product and an Eclipse application product will be established by tying the documentation to the .product file that is technically accounting for the Eclipse product.

The duty of the documentation tool is then to allocate all application model elements featured in this product and to process the artifacts in order to provide the raw input for further processing as presented in figure 5. At this point, a first impression will have been gathered on the elements in the application model, by the gaps that now become visible in the resulting documentation artifacts.

We currently conjecture the requirement to add the following model elements to the application model:

- Error Messages - bound to a command, formalizing errors occurring in the application
- Preferences - required as meta-information to the behaviour of certain commands
- Cheat-Sheets - used in Eclipse 3.x already to represent a sequence of steps, not yet available in the Eclipse application model

5.2 Integrating build and documentation generation

Documentation generation may be triggered at any time during the development process. The artifacts, however, have to take into account the product version they were compiled against. Ideally, the documentation code will be managed co-located with the application source code.

6 Scientific Approach

In order to find an answer to the research question as stated in section 4, a formal assessment method has to be determined and subsequently be applied and benchmarked against.

6.1 Determine assessment

Documentation quality is a very subjective matter, as it is strongly dependent on the previous knowledge of the application’s end user. A meaningful statement on the quality of such a documentation generation solution can hence only be found by statistically significant usage in different projects. This is clearly not tractable by the given project.

ISO/IEC 26514 [5][Annex G, pages 127 - 140] features a section called Requirements clauses and checklist for documentation products. Here 101 clauses are given which can be used to check the documentation applicability and conformance against.

6.2 Assess the solution

We will, after developing a prototype of the documentation generation system, select a set of about four Eclipse RCP based applications with a focus on optimized distribution with respect to fig. 7. This task might not be that trivial, as there do not exist a lot of open applications based on Eclipse 4 yet.

We will then trial the prototype on these applications, and benchmark the resulting documentation against the requirements clauses.

7 Conclusion

Harnessing the power of automated documentation generation based on the Eclipse application model would allow for a significant boost in development time and provide a novel approach combining software and documentation generation.
development. It could also rise software quality, as a con-
stantly up-to-date application documentation could be used
as a means to communicate the customers resp. users re-
quirements.

This paper outlines a novel approach for automated
application documentation generation, rendered possible
by the availability of a comprehensive application model. It
depicts the approach of harnessing the information avail-
able in the application model for automated documentation
generation, the open technical and research questions in do-
ing so. Potential shortcomings to the application model are
proposed, and will be substantiated in due course.

Acknowledgement

This work is partially sponsored the by The Austrian Re-
search Promotion Agency (FFG) under project number
840165.

References

[1] Donald E. Knuth, Literate Programming, The Com-
puter Journal, 27 (2), 1984, 97 - 111.

[2] W. L. Johnson, Dynamic (Re)Generation of Soft-
ware Documentation, Proc. 4th Systems Reengineering
Technology Workshop, J. Hopkins University, 1994

ware Specifications, Proceedings of the 3rd Australian
Document Computing Symposium (ADCS), 1998

[4] D. Amalfitano et al., Using dynamic analysis for gen-
erating end user documentation for Web 2.0 applica-
tions, 13th IEEE International Symposium on Web Sys-
tems Evolution (WSE 2011), Williamsburg, VA, USA,
September 30, 2011

[5] ISO/IEC 26514, Systems and software engineering -
Requirements for designers and developers of user
documentation, International Standard, First Edition,

To Guide For Project Staff (CreateSpace Independent
Publishing Platform, 2011)

[7] IEEE Standard for Software User Documentation In-
ternational Standard, Revision of IEEE Std 1063-

[8] G. Goth, Beware the March of this IDE: Eclipse is
overshadowing the other tool technologies, IEEE Soft-
ware vol. 22, no. 4, 108 - 111, 2005

(Amsterdam, Addison-Wesley Longman, 2008)

[10] L. Vogel, Eclipse 4 Application Development: The
complete guide to Eclipse 4 RCP development (Vol. 1)
(2012)

neering - Developing user documentation in an agile
environment, International Standard, Corrected Ver-

signing, Coding, and Packaging Java Applications
(Amsterdam, Addison Wesley, 2011)