

26. Documentation as Synchronized Dependent Model in a Macromodel

Documentation Generation as App for RAG

Prof. Dr. rer. nat. Uwe Aßmann

Institut für Software- und
Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

TU Dresden

<http://st.inf.tu-dresden.de>

Version 19-0.4, 07.12.19

- 1) Tasks
- 2) Template-Driven
Documentation Tools
- 3) Literate Programming
- 4) Elucidative Modeling and
Documentation Tools



DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

- ▶ C. Wilke, A. Bartho, J. Schroeter, S. Karol, U. Aßmann. Elucidative Development for Model-Based Documentation and Language Specification (Extended Version). Technische Universität Dresden. Institut für Software- und Multimediatechnik. Technical Reports TUD-FI12-01-Januar 2012, ISSN 1430-211X.
 - <http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-83442>
- ▶ Andreas Bartho. Elucidative Modeling. PhD thesis, Technische Universität Dresden, Fakultät Informatik, May 2014.
 - <http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-208060>
 - <https://www.linkedin.com/pub/andreas-bartho/ba/922/8a4?trk=pub-pbmap>



26.1 Tasks of Documentation Tools

http://en.wikipedia.org/wiki/Software_documentation



Basics of Software Documentation

4

Model-Driven Software Development in Technical Spaces (MOST)

- ▶ Documentation is a means of **communication**
 - between developers and future developers
 - between coders and testers
 - between developers and managers (for reviews and audits)
- ▶ Problems:
 - Documentation *ages* because code is modified and evolved
 - Good documentation costs time and money
- ▶ Different kinds of documentation:
 - **Generated documentation** is derived from code and models
 - **Integrated Documentation** is derived from the code (e.g., in comments), e.g., JavaDoc
 - **Elucidative Documentation**, derives both from another and keeps it consistent (generative or round-trip engineering)
- ▶ Standards:
 - national DIN 66230, 66231, 66232, 66270(1998)
 - international ISO/IEC 6592(2000), ISO/IEC 18019(2004)

Without documentation, a program is not software

Quelle: [24 S. 241 ff.]

Taxonomy of Documentation Documents

- ▶ **User documentation** (Benutzerdokumentation) explains the program to end users
 - Tutorials, user handbook, online documentation
- ▶ **System documentation** for installation, test cases, code documentation, maintenance, operations
 - **API documentation** documents interfaces of the system or framework, to let programmers use them for writing apps
- ▶ **Project documentation**
 - Developer documentation
 - Project documentation (project plan, requirements specification, status reports, after study)
- ▶ **Quality documentation**
 - Test-, review, audit documentation
- ▶ **Process documentation**
 - Standards, processes

Quelle: [24 S. 245 ff.]

Tasks of Documentation Tools

- ▶ Basically, documentation generation is similar to code generation. Documentation is created in higher-order attributes on a link tree by a RAG
- ▶ **Documentation generation is an app of a RAG**
- ▶ **Generation** of derived documents from code and models
 - Generation of Word, odt, rtf, xml, html formats
 - Generation of figures (svg, png, pdf)
 - Generation of snippets and generic snippets
 - Back-linking to originals
- ▶ **Filling** of documentation templates (hedge-principle)
- ▶ **Parameterization** with layouts
 - via css-style sheets

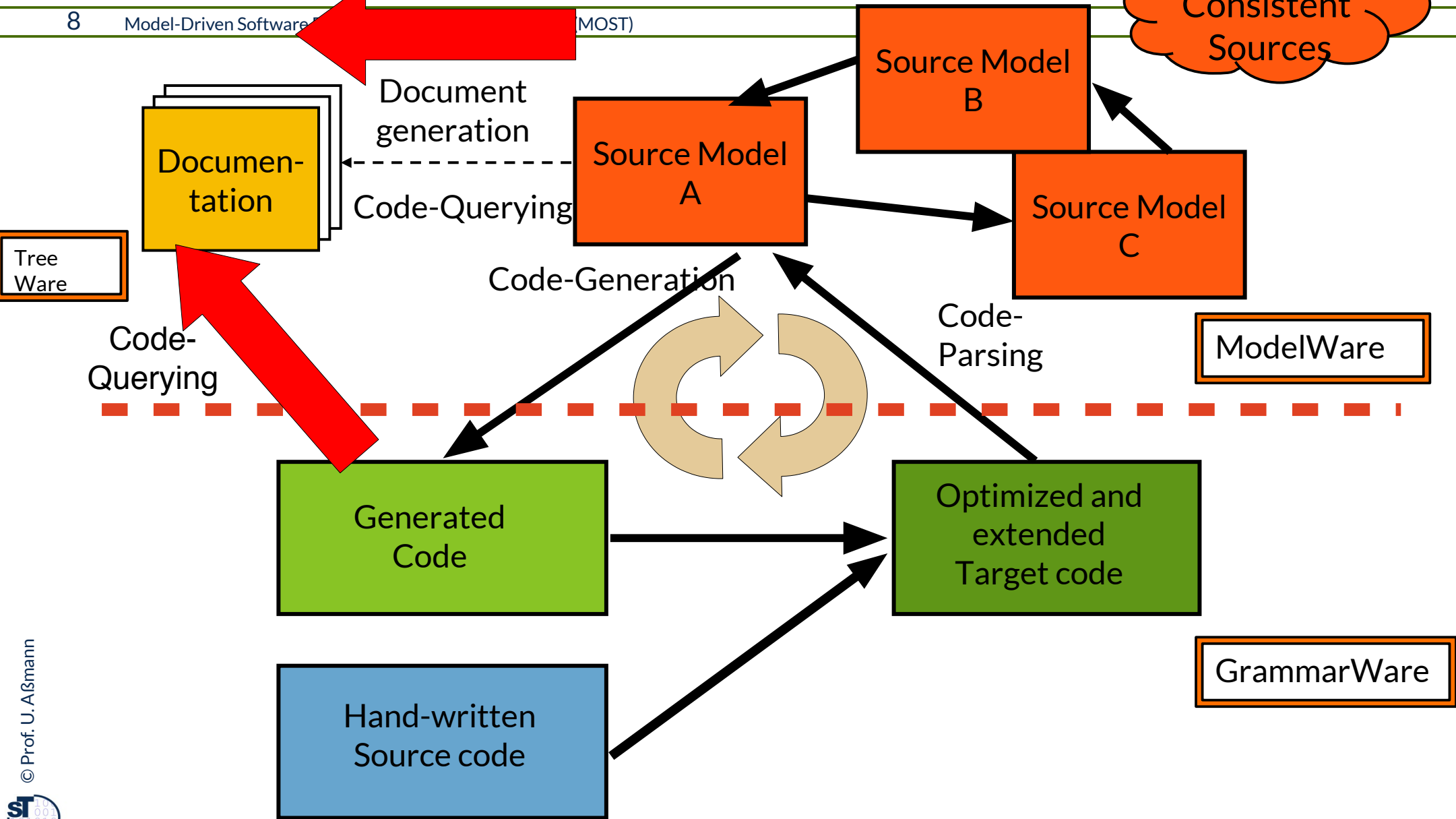


26.2 Generative, Template-Driven Documentation Tools

.. Documentation derived from code and models, based on template-based code generation



Macromodel Principle and Round-Trip Engineering



Documentation Tool JavaDoc is a Template Expander

- ▶ JavaDoc reads Java source code and extracts html from the code comments, based on **html templates**
 - Typical hedge-based code generation with generic snippets
- ▶ Generation of additional contents and indices
- ▶ Controlled by Java metadata attributes
 - @author, @date, @param
- ▶ Layouting via plugin classes called *doclets*
- ▶ JavaDoc has been realized for all programming languages

JavaDoc is a Typical HRAG Application

- ▶ The html documentation is computed in a higher-order synthesized attribute
`htmlDoc : HTML`

```
// schematic, synthesis from bottom to top
Interpretation javaDoc(Tree → Tree) {
  Attributions of Root(classes[]) {
    this.htmlDoc := map + classes.htmlDoc;
    <println(„Result is %S“, this.htmlDoc)>
  }
  Attributions of Class(superclass:Class,methods{}) {
    this.htmlDoc := <superclass.Name + st2.htmlDoc;
  }
  Attributions of Method(name,comment) {
    this.htmlDoc := „<h1>“+name+“</h2>“+comment.htmlDoc;
  }
  Attributions of Comment(text) {
    this.htmlDoc := text;
  }
}
```

Composition of Separated Hand-Written and Generated Documentation Snippets

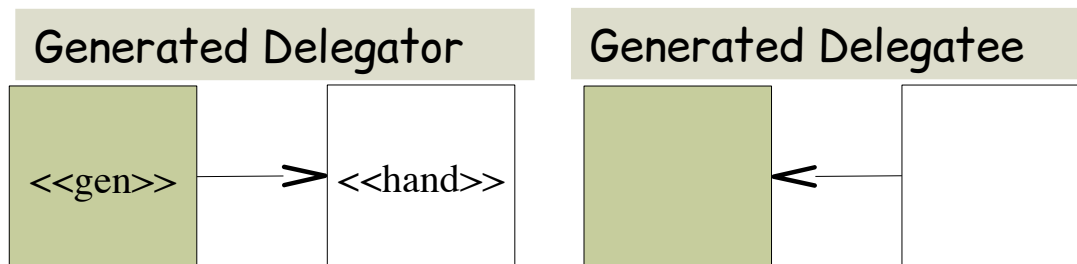
11

Model-Driven Software Development in Technical Spaces (MOST)

- ▶ **In separate files:** Coupling by “include”
- ▶ Only possible if document format supports subdocument inclusion
 - e.g., TeX or Framemaker

In one file:

Coupling with **hedges (Trennmarkierung)**



Generated Wrapper

```
/** Generated documentation  
***/
```

```
/** Hedge ***/
```

```
... Hand-written  
Documentation ...
```

```
/** Hedge ***/
```



26.3 Literate Programming

- ▶ They integrate code, models and documentation by **separating code from documentation**



How to Write Documentation and Tutorials?



The screenshot shows a Mozilla Firefox browser window with the address bar containing `http://kiwientkerner.de/tutorial/shop.html`. The page content is highlighted in yellow and contains the following Java code snippet:

```
public class SalesPointApplication extends Shop {  
  
    public SalesPointApplication() {  
        super();  
    }  
  
    public void start() {  
        super.start();  
        try {  
            Log.setGlobalOutputStream(new FileOutputStream("logfile.log", true));  
        }  
        catch (IOException ioex) {  
            System.err.println("Unable to create log file.");  
        }  
    }  
}
```

Verhalten beim Schließen

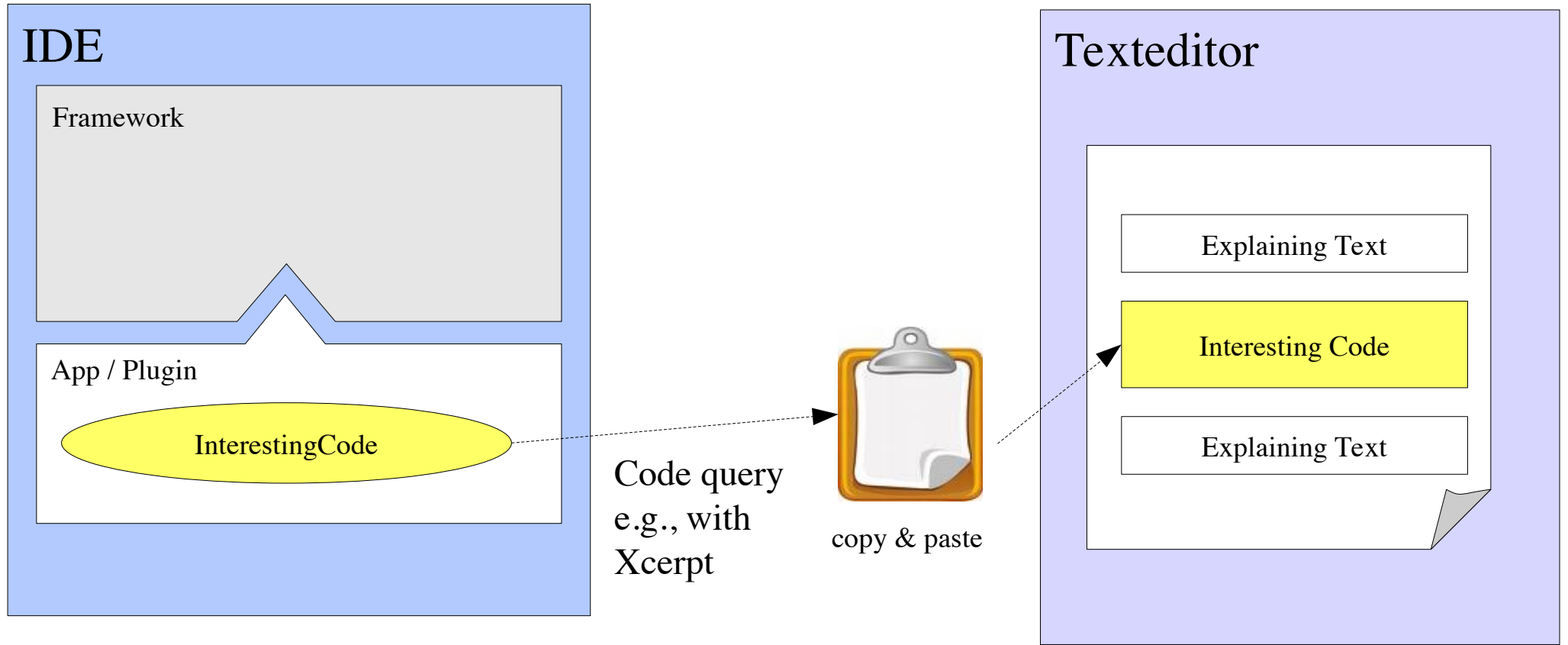
Wenn die Anwendung geschlossen wird, öffnet `SalesPoint` automatisch einen Speichern-Dialog um den aktuellen Status der Anwendung zu sichern. Wenn das nicht gewünscht wird lässt sich dieses Verhalten leicht ändern. Es ist die `quit()`-Methode des Shops zu überschreiben.

Mit `shutdown(boolean)` wird versucht, alle laufenden Prozesse zu beenden und das Shopfenster zu schließen. Hatte dies Erfolg wird `true` zurückgeliefert. Als Argument wird erwartet, ob der Speicherdialog erscheinen soll oder nicht. Es ist zu beachten, dass das Programm nach dem Schließen des Shops noch nicht beendet ist. Die `exit`-Methode ist manuell aufzurufen.

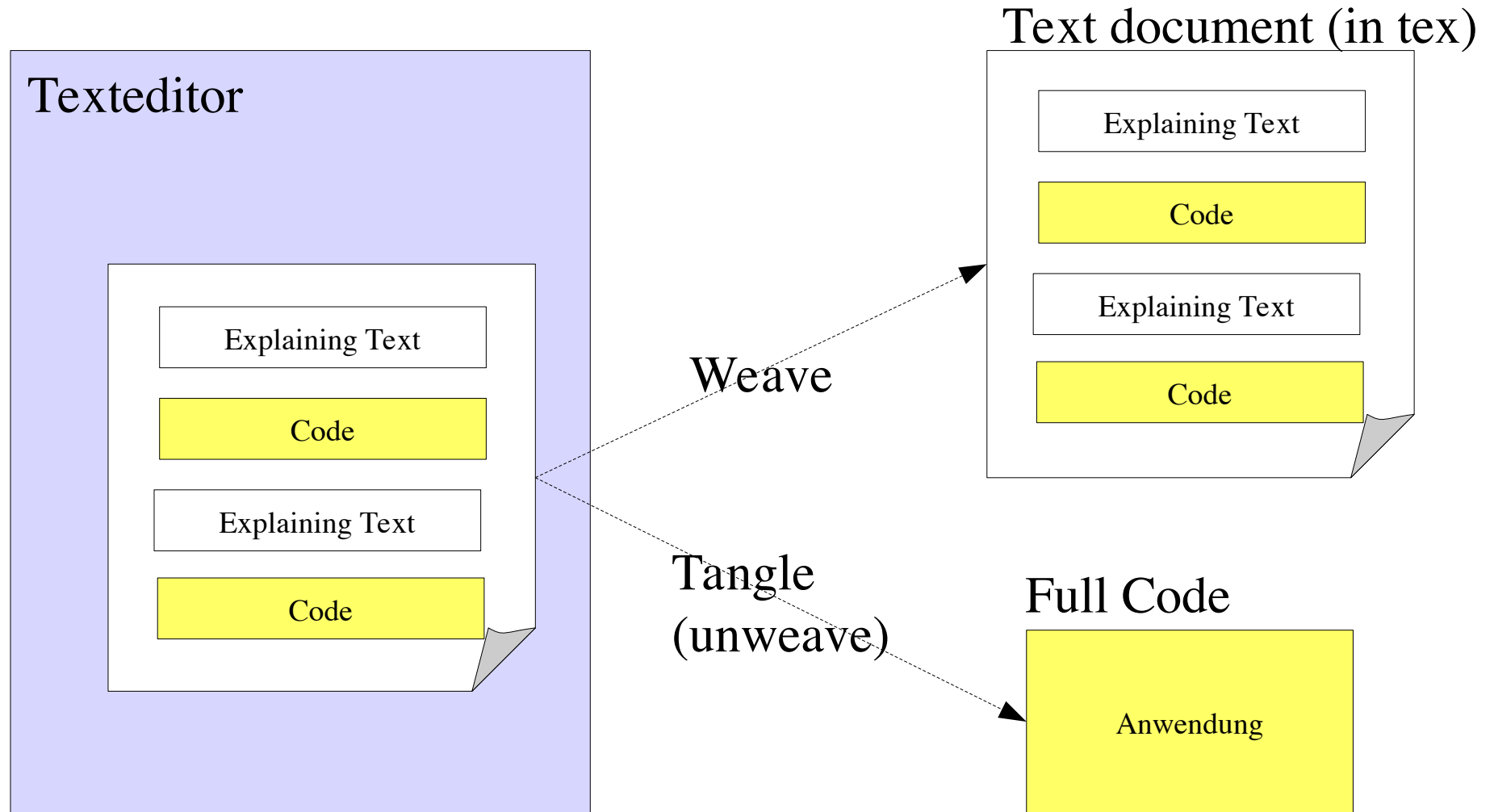
```
import sale.Shop;  
import log.Log;  
import java.io.*; //IOException, FileOutputStream  
  
public class SalesPointApplication extends Shop {  
  
    public SalesPointApplication() {  
        super();  
    }  
  
    public void start() {
```

At the bottom of the browser window, there are buttons for "Adblock" and "Fertig".

Manual Writing of Tutorials



Literate Programming by Code Unweaving



[[The program text below specifies the “expanded meaning” of ‘⟨Program to print . . . numbers 2⟩’; notice that it involves the top-level descriptions of three other sections. When those top-level descriptions are replaced by their expanded meanings, a syntactically correct PASCAL program will be obtained.]]

```
⟨Program to print the first thousand prime
  numbers 2⟩ ≡
program print_primes(output);
  const m = 1000;
  ⟨Other constants of the program 5⟩
  var ⟨Variables of the program 4⟩
  begin ⟨Print the first m prime numbers 3⟩;
  end.
```

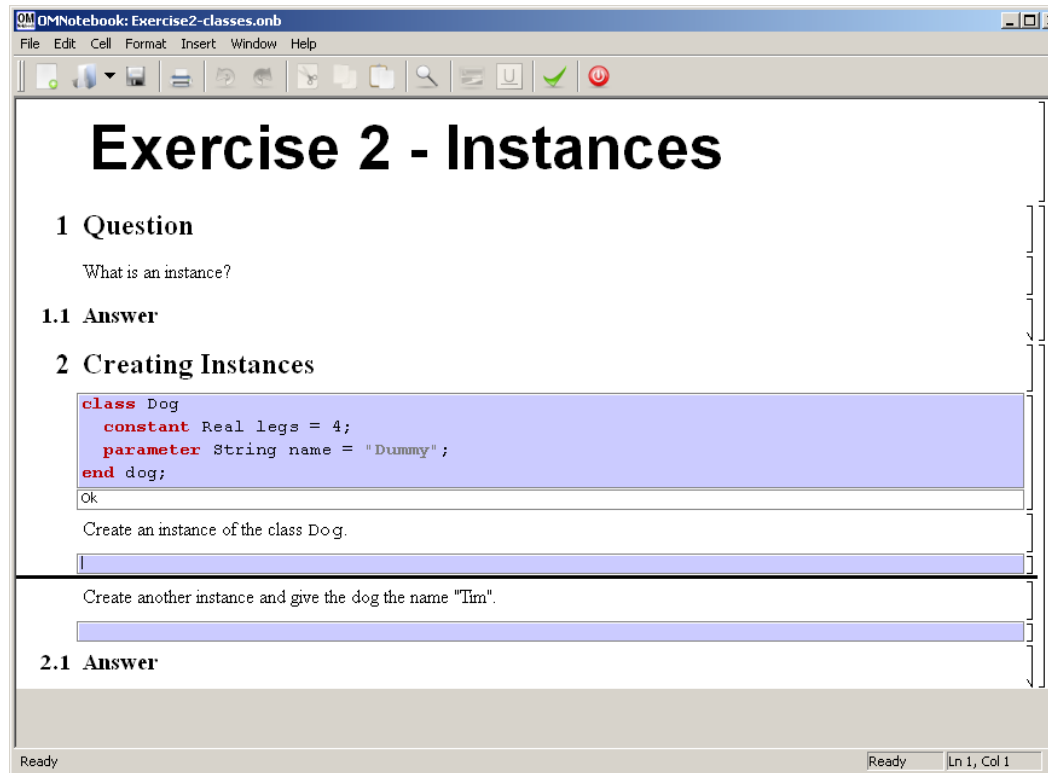
[Literate Programming
von Donald E. Knuth]

- ▶ The TeX engine is programmed literately
- ▶ Overview: <http://www.literateprogramming.com/>
- ▶ OMNotebook/DrModelica: <http://www.modelica.org/tools>

OMNotebook – Literate Programming with DrModelica

17

Model-Driven Software Development in Technical Spaces (MOST)



The screenshot shows the OMNotebook application window titled "OMNotebook: Exercise2-classes.onb". The window contains the following content:

Exercise 2 - Instances

1 Question

What is an instance?

1.1 Answer

2 Creating Instances

```
class Dog
  constant Real legs = 4;
  parameter String name = "Dummy";
end dog;
```

Ok

Create an instance of the class Dog.

Create another instance and give the dog the name "Tim".

2.1 Answer

The status bar at the bottom indicates "Ready" and "Ln 1, Col 1".

- ▶ Linked documents with interactive exercises
- ▶ Inspired by DrScheme und DrJava, learning tools for Scheme resp. Java
- ▶ www.openmodelica.org

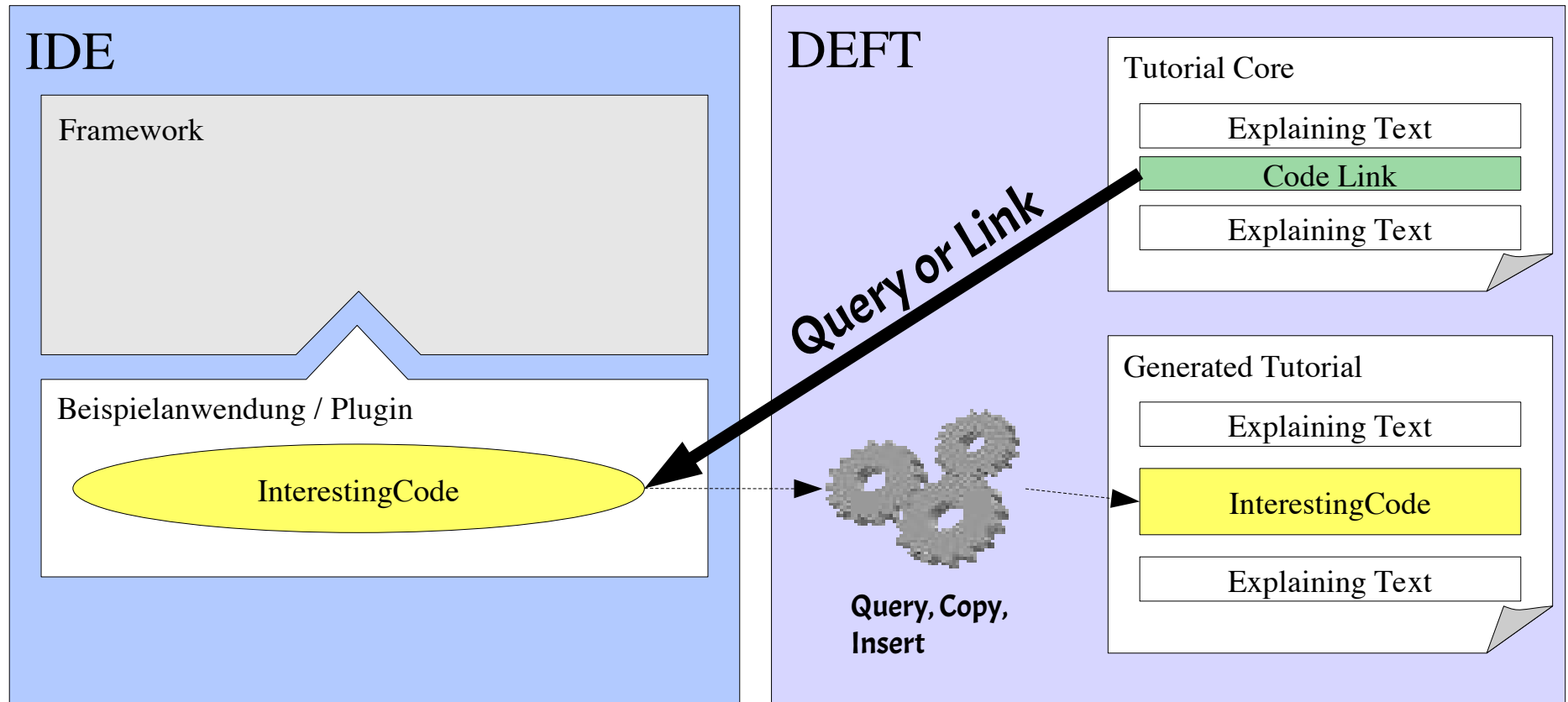


26.4 Elucidative Documentation Tools

- ▶ They link code, models and documentation by **model and code mapping**



Elucidative Programming Links Documentation with Queries to Code



Elucidative Programming

20

Model-Driven Software Development in Technical Spaces (MOST)

The screenshot displays the 'Scheme Elucidator' environment. At the top, there are navigation icons and a menu with 'time' and 'general' options. Below this, a navigation bar shows three sections: '1 Introduction', '2 The solution', and '3 Post Scriptum'. The 'Introduction' section is selected, and its content is shown in a blue-highlighted area on the left. The right side of the window shows the corresponding Scheme code, with some lines highlighted in yellow. A red dashed speech bubble points to the right side of the window, containing the text '„Scheme Elucidator“ Environment'.

1 Introduction
Generated: 22. Juni 2000, 10:23:01

2 The solution

3 Post Scriptum

1 Introduction
In this introductory section we first discuss time system formats and a function in Scheme which returns the current time. Then we discuss the issue of normalization and two possible ways to attack the problem.

1.1 Time systems and functions
1.2 The plan of attack

1.1 Time systems and functions
There are several different standards for representation of time on a Computer. *Universal*

```
;; Fixed second counts and calendar facts.  
(define seconds-in-a-normal-year 31536000)  
(define seconds-in-a-leap-year 31622400)  
(define seconds-in-a-week 604800)  
(define seconds-in-a-day 86400)  
(define seconds-in-an-hour 3600)  
(define base-year 1970)  
(define month-length-normal-year  
  (vector 31 28 31 30 31 30 31 31 30 31 30 31))
```

- ▶ Elucidative Programming shows documentation and code in parallel
- ▶ <http://www.cs.aau.dk/~normark/elucidative-programming/>
- ▶ <http://deftproject.org>

Development Environment For Tutorials (DEFT www.deftproject.org)

21

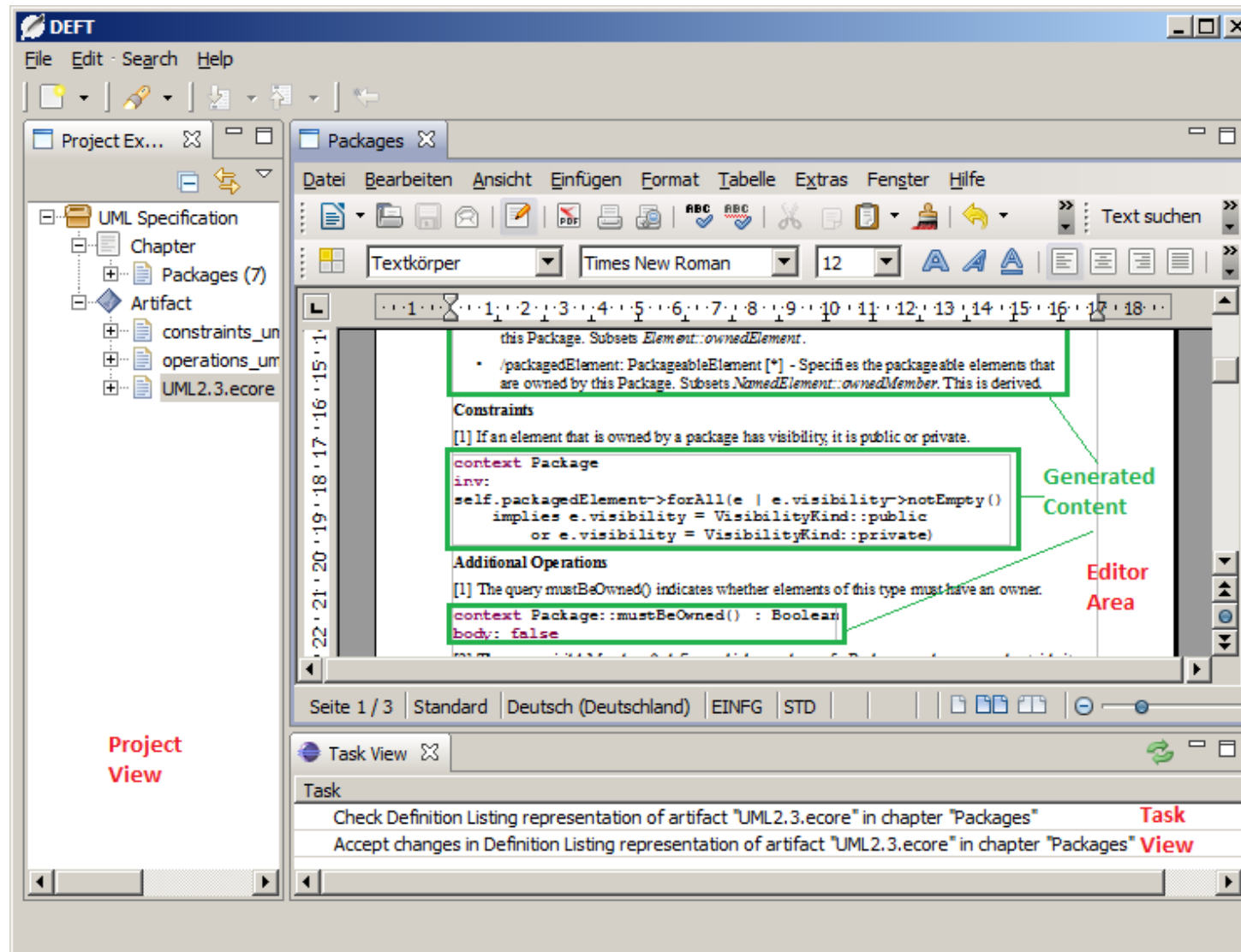
Model-Driven Software Development in Technical Spaces (MOST)

The screenshot displays the DEFT development environment with the following components:

- Project Explorer (left):** Shows a project named "Fahrkartenautomat" with a tree structure including "Chapters", "Code files", "Code snippets", "Images", and "Tutorials". The "Code files" folder is expanded to show several .cs files.
- Texteditor (center):** Displays the source code for "FahrkartenAutomat.cs". A yellow highlight covers a `try` block that opens a file stream for logging. A red label "Eingebettetes Codefragment" points to this block. Below the code, a paragraph explains the logging process, mentioning `FileStream fs` and `Log.Log`.
- Editor Outline (top right):** Shows a simple tree view with "Start" and "Start der Anwendung". A red label "Kapitelstruktur" is placed below it.
- Code Outline (bottom right):** Shows a detailed AST (Abstract Syntax Tree) for the project, including nodes for "UsingDirectives", "FahrkartenAutomat", "TICKETS", "MONEY", "DefaultMenuSheet", "FahrkartenAutomat(string)", "createDisplayManager()", "Main(string[])", "FahrkarteVerkaufenAction", and "WartungAction". A red label "AST-Fenster" is placed below it.

Additional red labels are present: "Projektübersicht" at the bottom left and "Texteditor" at the bottom center.

Embedding UML Constraints for UML Models into Documentation



Development Environment For Tutorials

- ▶ Eclipse RCP application, language independent
- ▶ Management of code, models and text
- ▶ Prettyprinting of code fragments from code templates
- ▶ Hot update of generated documentation
 - Automatic update of embedded code fragments
 - Notiviation if code fragments have changed

Start der Anwendung

In der Klasse `FahrkartenAutomat` befindet sich die `Main`-Methode, mit der sich das Programm starten lässt. Dort werden Daten initialisiert und der `FahrkartenAutomat` instanziiert.

Logging

Der erste Schritt ist die Konfiguration des Loggings. Das `SalesPoint`-Framework bietet Funktionen und Datentypen an, mit denen Aktionen geloggt werden können. Es gibt GUI-Komponenten, mit denen die Inhalte des Logs wieder nutzerfreundlich angezeigt werden können. Eine Anzeige des Logs ist derzeit nicht im `FahrkartenAutomaten` implementiert, geloggt wird aber trotzdem schon.

Um Logging zu verwenden, muss ein Stream auf die Logdatei geöffnet werden.

```
try
{
    FileStream fs = new FileStream("test.log", FileMode.Create);
    Log.Log.GlobalOutputStream = new System.IO.BinaryWriter(fs);
}
catch (IOException)
{
    System.Console.Error.WriteLine("Unable to create Log file.");
    return ;
}
```

Es wird versucht, die Datei `test.log` zu erstellen. Falls sie schon existiert, wird sie überschrieben. Der `FileStream fs` kann Daten (Bytes) vom Programm in die Datei schreiben. Byteweises Schreiben von Informationen ist allerdings sehr umständlich. Ein `BinaryWriter` kapselt den `FileStream` und bietet Methoden zum Schreiben von Strings, Zahlen, und Anderem. Der globale Log-Klasse der Anwendung, `Log.Log`, wird dieser `BinaryWriter` zugewiesen. Alle

```
{
}

protected override DisplayManager createDisplayManager()
{
    Size d = System.Windows.Forms.Screen.PrimaryScreen.Bounds.Size;
    Point tempAux = new Point((d.Width - 100) / 2, (d.Height - 80) / 2);
    Point tempAux2 = new Point(5, 5);
    return new AWTDisplayManager(this, ref tempAux, ref tempAux2);
}

[STAThread]
public static void Main(string[] args)
{
    //System initialisieren
    try
    {
        FileStream fs = new FileStream("test.log", FileMode.Create);
        Log.Log.GlobalOutputStream = new System.IO.BinaryWriter(fs);
    }
    catch (IOException)
    {
        System.Console.Error.WriteLine("Unable to create Log file.");
        return ;
    }

    // Kataloge anlegen

    // Fahrscheinkatalog
    Catalog cTickets = Catalog.forName(TICKETS);

    cTickets.addItem(new Fahrschein("Einzelfahrt", 300));
    cTickets.addItem(new Fahrschein("Sammelfahrschein", 1500));
    cTickets.addItem(new Fahrschein("ermäßigte Einzelfahrt", 150));
}
```



The End

- ▶ Why is generation of documentation similar to code generation?
- ▶ Explain why a higher-order RAG is useful for documentation generation
- ▶ Which role does a pattern-matching language such as Xcerpt play in documentation generation?
- ▶ Why is the generation of documentation part of a macromodel?
- ▶ Why is a documentation a *derived model*?
- ▶ What happens if text from the API documentation flows back into the code as comments?



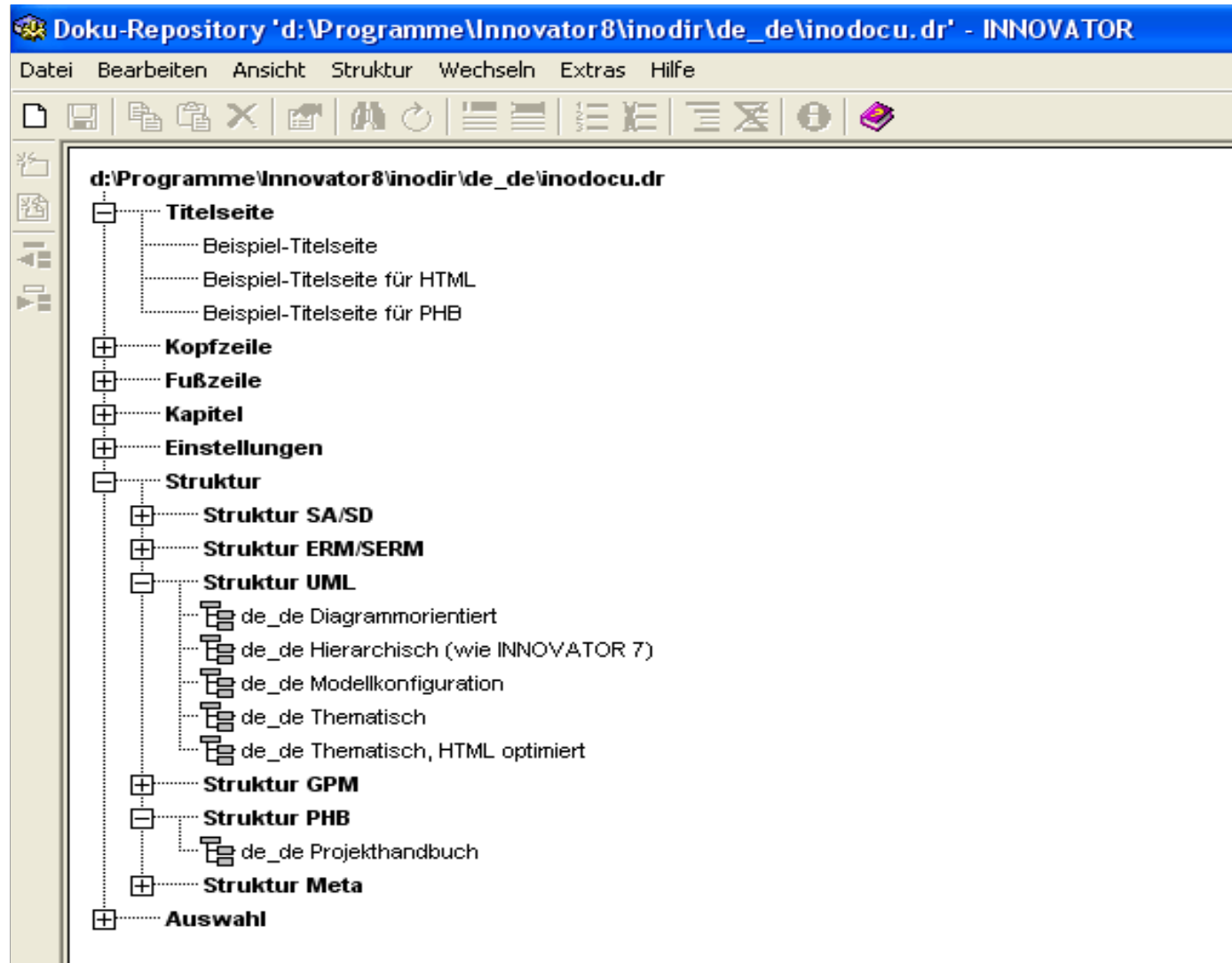
A.1 Other Template Expanders for Documentation Generation



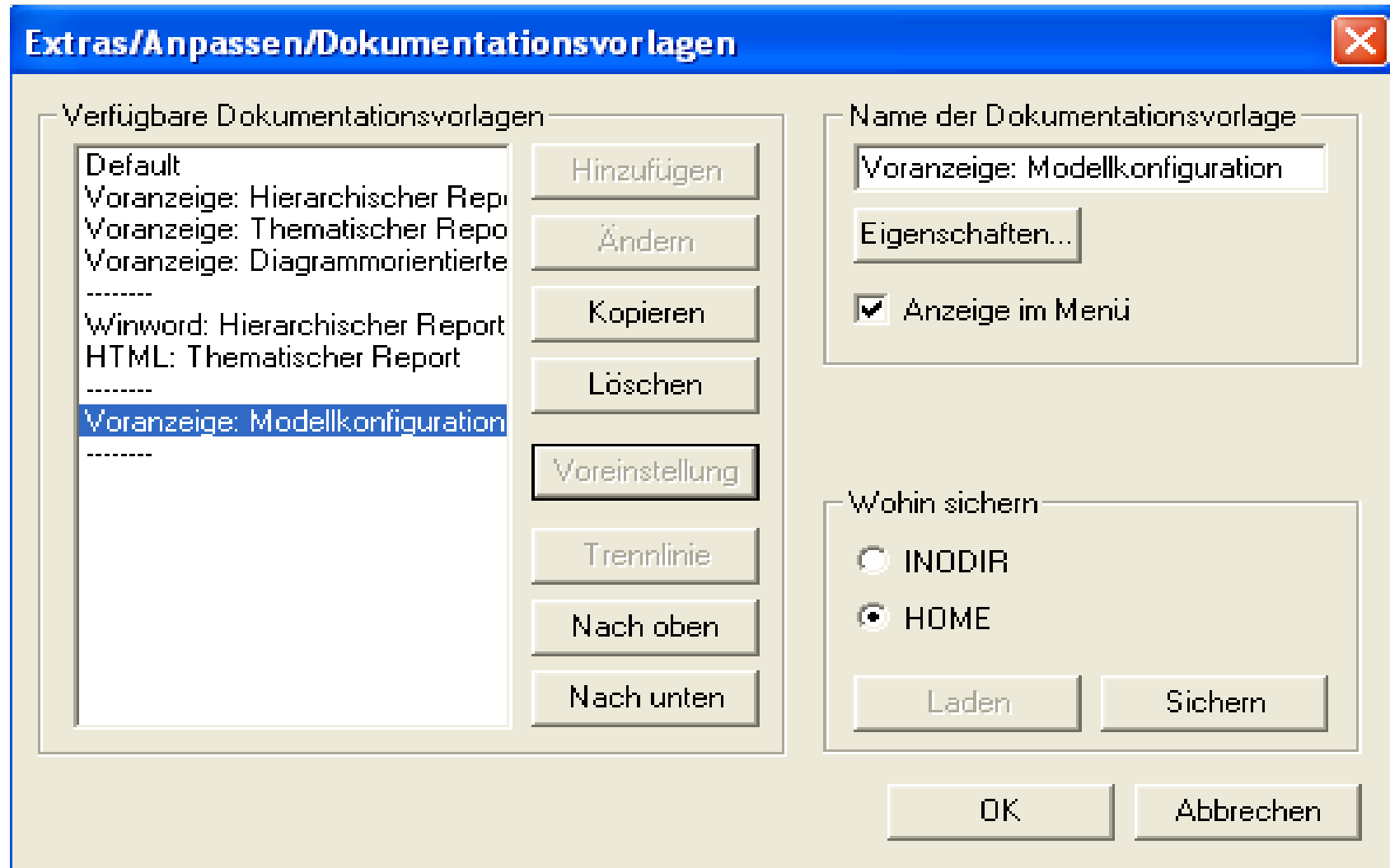
Documentation Tools of MID Innovator

- ▶ Innovator provides documentation templates, into which diagrams, models, code can be embedded
- ▶ Several formats:
 - pdf
 - Word
 - ASCII
 - XML

Ex.: Innovator Documentation Template (Dokumentationsvorlage)



Ex.: Innovator Documentation Template (Dokumentationsvorlage): Adaptation



Innovator - Generated Example Word Document

31 Model-Driven Software Development in Technical Spaces (MOST)

Voranzeige c:\temp\ldr21912

Datei Wechseln Optionen Hilfe

- i -

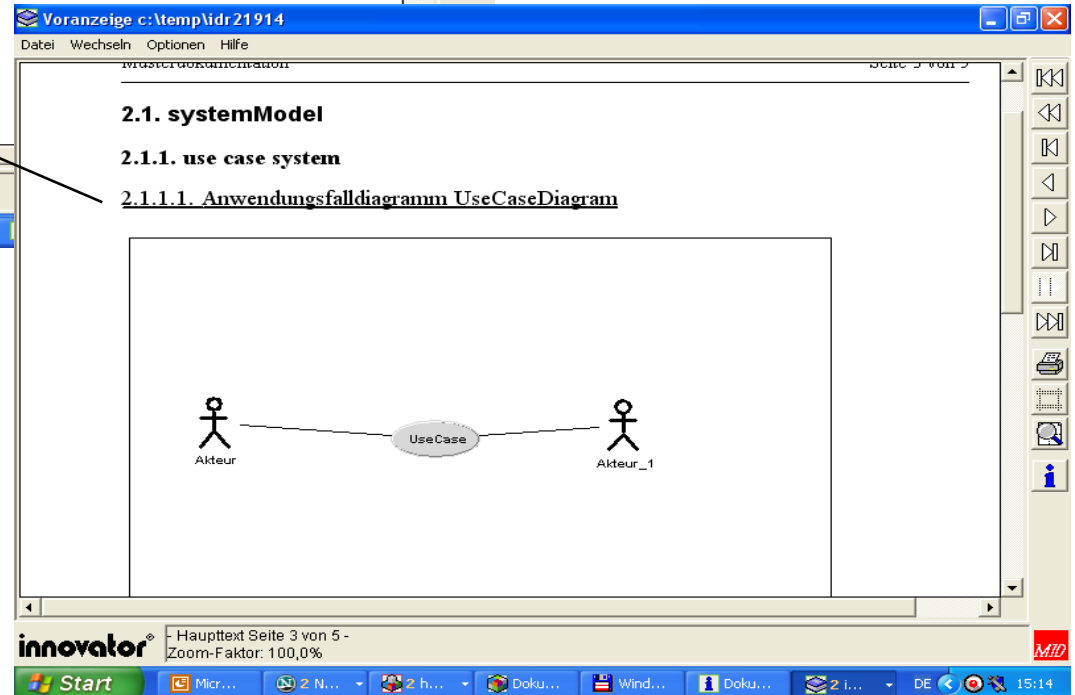
Inhaltsverzeichnis

1. externes Kapitel	1
1.1. Unterkapitel1	1
1.2. Unterkapitel2	1
1.3. Unterkapitel3	1
2. Doku	2
2.1. systemModel	3
2.1.1. use case system	3
2.1.1.1. Anwendungsfalldiagramm UseCaseDiagram	3
2.1.1.2. Paketdiagramm Create Defaults for Use Cases	4
3. Index	5

innovator® - Inhaltsverzeichnis Seite 1 von 1 -
Zoom-Faktor: 100,0%

Start | Micr... | 2 N... | 2 h... | Doku... | Wind...

Table of Contents



Integration of a Use Case Diagram
(section 2.1.1.1.)

Index is generated