

Transformational Abstraction for Java (TAJ)

Advisor: Dr. Spencer Rugaber

Student: Sergio Berzosa González

Motivation

A major problem in software maintenance and reverse engineering is the lack of documentation that represents the actual state of the application source code. When a software product needs to be updated to fix bugs, add new functionality, or ported to other system, usually developers find themselves with documentation that does not reflect the actual state of the application (and the source code) making it hard to comprehend how the code accomplishes the functionality stated in the different high level documentation elements. Developers then must read and understand the source code, using code comments as the low level documentation, which may themselves be out of date.

TAJ's functionality is based on the idea that the source code is created to simulate elements pertaining to a well define universe (the problem domain). The source code is driven by the problem domain, and thus source code constructions are related to elements contained in the problem domain. Existing tools provide the ability to document programs at low levels (source code comments) or at high levels of abstraction (architecture diagrams), but the creation of the source code is affected by low level design decisions that obscure the higher level abstractions. TAJ fills this gap by explicitly linking source code constructions to elements in the problem domain.

Traditional documentation allows developers to get information about different levels of abstraction, from the simple source code comments, to class diagrams, state charts, sequence diagrams, etc. Those different types of documentation are not directly related to each other, so making use of those documents means having to manually fill the gap between the different representations.

In a sense TAJ offers an integral solution by allowing users to create and view documentations at multiple abstraction levels while maintaining a relation between the different levels. By offering these levels of abstraction different kind of users can obtain just the necessary information they need, from a view where the low level is represented to a high level where the main interactions between the different components can be easily observed. Moreover, users can interactively drill down or move up in the abstraction levels at different parts of the source code file, so they can always get the most appropriate representation for each part of the source code.

All these elements constitute a system that helps developers in two different scenarios:

- **Building up an understanding from undocumented code:** By following the process of examining the code looking for domain elements' implementations, and by modifying the domain to more precisely represent the code, developers can build up understanding of the system.
- **Using TAJ to understand previously annotated code:** When a new developer needs to start working on code created by other developer, TAJ can be used as a documentation element to detect which sections of the source code are used to implement a certain domain element. This would allow a quick navigation to the source code points of interests to build a general understanding of the application.

Architecture

TAJ can be viewed as two different components that interact with each other to create a mapping between source code and the domain.

TAJ editor: the editor allows developers to group continuous lines of code, giving it a brief description of the intent of the grouped code; these elements receive the name of chunks. The chunks can then be folded or unfolded to toggle between the source code and the description associated with the chunk. The chunk structure also allows grouping a contiguous list of source code lines or chunks, creating a tree-like structure.

TAJ domain editor: the domain editor component allows developers to create and modify a domain model. The different elements in the domain model can then be linked to different chunks created in the editor component of TAJ. The domain representation allows the creation of associations and specializations between the elements of the domain as well as the definition of operations and attributes for the different domain elements.

These elements are developed under the Eclipse framework to be integrated into the Eclipse application as a plug-in. The use of this popular IDE allows developers to easily integrate TAJ into their workflow.

Initial project status

Last version available of TAJ previous to release 2.0 was designed using the Eclipse framework 2.x. The release only included the TAJ editor as part of the plug-in, while the TAJ modeler was at a conceptual stage.

The TAJ editor enabled users to collapse contiguous lines of text into chunks. The structure generated by those folding could be saved in order to be later recovered for further modifications. Along with the java file being edited, TAJ created an addition file on the file system containing information of the different chunks created by the user. That filename was created using the filename of the java file appended with the string “.taj”

The process of saving the chunks could be considered destructive since the creation of chunks modified the original content of the java file being edited. Chunks that were collapsed, replaced the actual portions of the java code with the description associated with each of the chunks. This made it impossible to manipulate the java file without first unfolding the collapsed chunks to recover the original source code file.

Along the editor, an outline associated with it was created to show the different elements of the document being edited as a series of lines and chunks. This outline gives visual feedback of the hierarchy of the different chunks in the document.

Conversion to Eclipse Framework 3.x

After understanding how the last version of the plug-in was constructed and how it interacted with the Eclipse Framework, the next task was to be developed was porting the latest version of TAJ at that time, to the current version of the Eclipse Framework. This task involved the study of the modifications suffered by the Eclipse Framework from release 2 to release 3, how the TAJ source code could be modified to be compatible with it, and how the new features could be used to enhance the TAJ project.

Eclipse 3 includes a tool that allows developers to automatically modify an existing plug-code to run in the new Eclipse 3 framework. This is done by including a compatibility layer that enables legacy plug-ins to continue to use the same interfaces and classes by maintaining binary compatibility. After we were able to execute the plug-in using Eclipse 3, the task of modifying the TAJ source code so that it runs natively started. There are some exceptions where the API changes could not be done in any compatibility way. Fortunately, TAJ did not use any of those features, and thus we were able to easily get the code running using the previously mentioned compatibility layer.

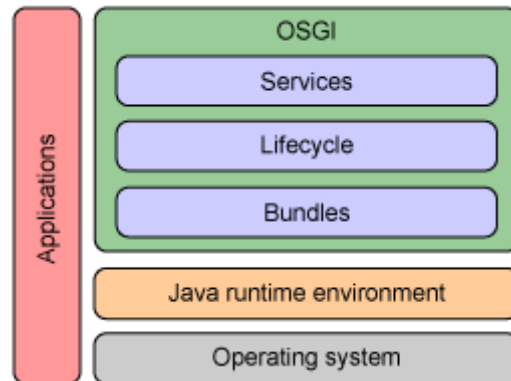
Removing the compatibility layer

The changes needed for removing the compatibility layer and running using native Eclipse 3 methodologies. Some of the changes suffered by the framework included the creation of new packages that. Some packages were moved between releases while some others were divided between various plug-ins.

For being able to compile the source code removing the compatibility layer new packages had to be imported so that the elements contained in the moved and split packages could be referenced correctly. Porting the code also involved revising the plug-in lifecycle model used by Eclipse, moving from a proprietary technology to a model driven by the OSGi specification.

OSGi specification and Eclipse

Starting with Eclipse 3.0 the runtime is based on the Open Services Gateway Initiative (OSGi). Previous to Eclipse 3.0 the framework used its own proprietary plug-in system to manage the installed plug-ins. As more plug-ins and requirements were incorporated into Eclipse IDE, the developers decided to substitute the proprietary model with OSGi. The OSGi defines the concept of bundle as collection of types and resources and associated inter-bundle prerequisite information that contributes to the system. OSGi also defines an infrastructure for a bundle's lifecycle and how bundles interact with each other.



Interaction of the layers in the host O.S, Java and OSGi

Eclipse implements a subset of OSGi centered in the modularization and lifecycle portion of the specification. However, it makes minimal use of the service support provided by OSGi. Instead, Eclipse keeps its own extension points that enable the bundle interaction.

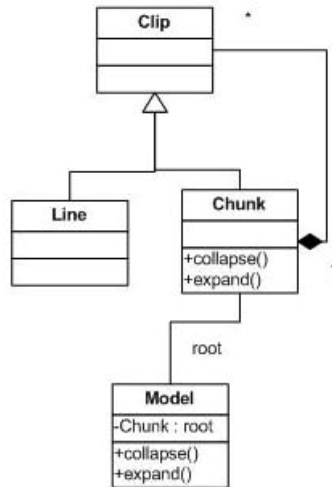
One of the key benefits of OSGi over the previous technology used by Eclipse is the ability to discover, load and upload bundles at run-time without the need of restarting the Eclipse IDE, and also to broadcast related events (eg: install, stop, uninstall...) to interested parties. In previous releases plug-ins were discovered during the Eclipse initialization and any modification to the plug-ins needed a restart Eclipse to take effect. The new model also encourages doing initialization in a classic lazy style, where the data structures and models initialization is deferred until it is actually needed, and not when the plug-in/bundle is activated.

A problem encountered when adapting the plug-in was importing a package in the source code without defining the import in the corresponding area of the manifest file. The source code compiled without error, but when executing the plug-in it would fail when opening a file because the source could not be able to load a certain class. By explicitly importing the package in the OSGi manifest file the class was available for use by the plug-in.

The current OSGi implementation used in Eclipse is called Equinox and is based on the OSGi R4 core framework specification.

Folding

The TAJ model created for the different code abstractions constructed during the tool usage create a tree structure that contains, lines of code as well as chunks. The diagram below shows the classes used to create this structure.



Class diagram of the TAJ model

The diagram shows that we have implemented the Composite design pattern. There is a class called *Clip* from which the classes *Line* and *Chunk* inherit. The class *Line* represents each one of the lines of the source code document. When a section of code is collapsed by the user a new *Chunk* is created, since the class *Chunk* is an aggregation of clips, a chunk is allowed to contain not only lines, but also other chunks.

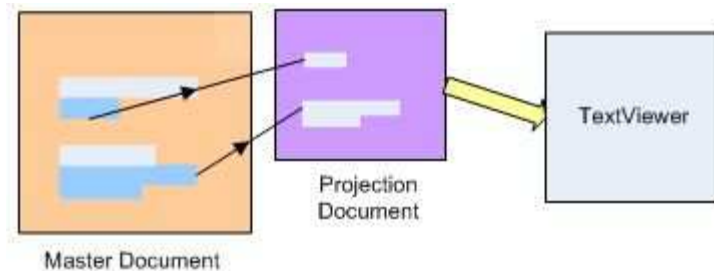
To represent the whole structure there is a model that contains an attribute that references the main chunk. Since we only need one reference to the whole structure, the design is simplified. When a new model is created the root element of the model is a chunk that includes all the lines of the document.

Eclipse integration

When it was time to implement the folding in Eclipse we had the choice to create our own infrastructure using functionality provided by Eclipse or to make use of the folding included in the Eclipse framework. The folding, as it is implemented by eclipse, is used to hide/show portions of text based on an analysis of the text, creating different sections that can then manipulated making use of UI elements provided by the platform.

The decision taken was to make use of these folding capabilities of the framework instead of implementing our own solution. This way we could use an already existing component that provided the basic functionality that we needed.

The folding in eclipse makes use of three main classes, those classes create a pipeline structure than transforms the original text of the document into the view that is shown to the user.



Eclipse folding implementation

Since this implementation only support the display of text contained in the master document, the descriptions associated with each chunk have to be inserted in the master document so that they can be displayed in the text viewer. It is important to keep the chunks unaware of this issue, so that the initial and final line numbers of the chunks refer to the original document. In order to do this, the editor has to compute the difference between the number of the displayed lines and that same line in the original document. This value is not constant and varies from chunk to chunk depending on the chunk position in the model structure.

The decision of keeping the chunks unaware of this offset would allow us to modify the implementation without altering the Chunk code.

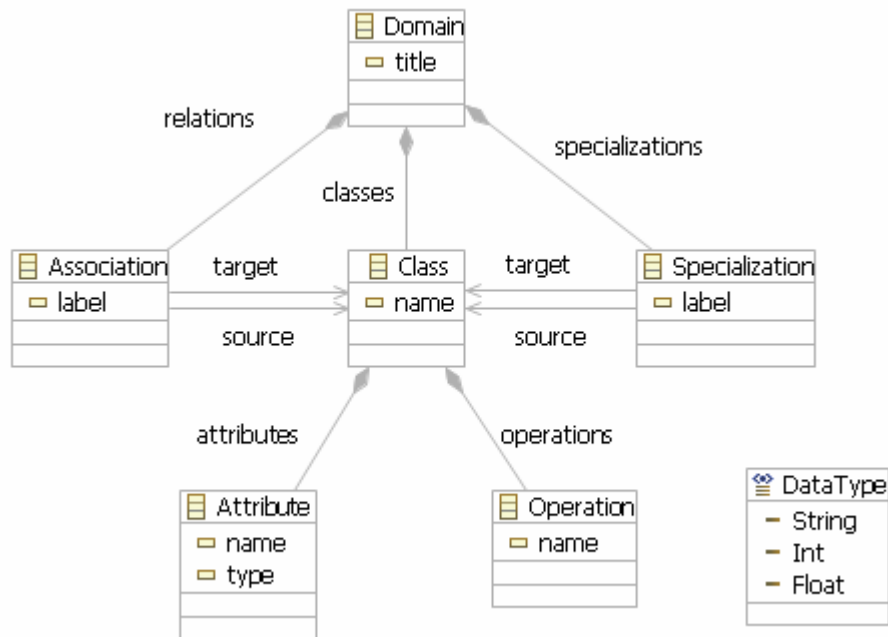
In order to create the different sections that can be collapse we have to make use of the projection annotations, these annotations tell eclipse which sections of the document are collapsible.

Here, the Observer pattern was used. The TajEditor object subscribes to the document model. When the user request an action that modifies the number of chunks in the document (operations of collapse or expand), the editor calls the appropriate method in the model object. Once the new chunks have been created, the model notifies to all the observer of that action so they can take the appropriate measures.

When the editor is notified of a change on the structure of the document, it creates or removes the appropriate projection annotation so that the view shown to the users is synchronized with the TAJ structure of the document. The framework automatically creates all the UI components and adds the functionality based on the annotations.

TAJ Domain Editor

The domain editor implemented allows users to create a domain to be linked to the different chunks created in the TAJ editor. A domain consists of a series of classes, identified by their name, which can contain attributes and operations. Those different classes can be related by using associations or specializations.



Domain model

GMF

The model editor makes use of the Graphical Modeling Framework (GMF) in order to create a graphical editor for an underlying model definition. GMF provides a runtime infrastructure for developing graphical editors based on the Eclipse Modeling Framework Project (EMF) and the Graphical Editing Framework (GEF).

By making use of GMF we are able to create a complete editor that supports all the elements defined by the underlying model defined by EMF. At the same time, GMF makes use of GEF to generate a rich graphical editor. The framework requires the creation of different elements that once combined contribute to the creation of the final editor.

Graphical Definition

A graphical definition model is needed for defining how the different elements of the domain will be represented in the editor. In the case of TAJ this graphical definition includes rectangular figures for the classes, compartments for the

attributes and operations of the classes, lines for the associations and specializations, etc.

Tooling Definition

The tooling definition is needed so GMF knows which tools it has to place in the palette situated that will be situated at the tight side of the domain model being created. Usually this definition includes the different elements that can me added to the model. By default this palette also includes some common elements not associated with a particular model, but can be useful for the users (eg: create a note)

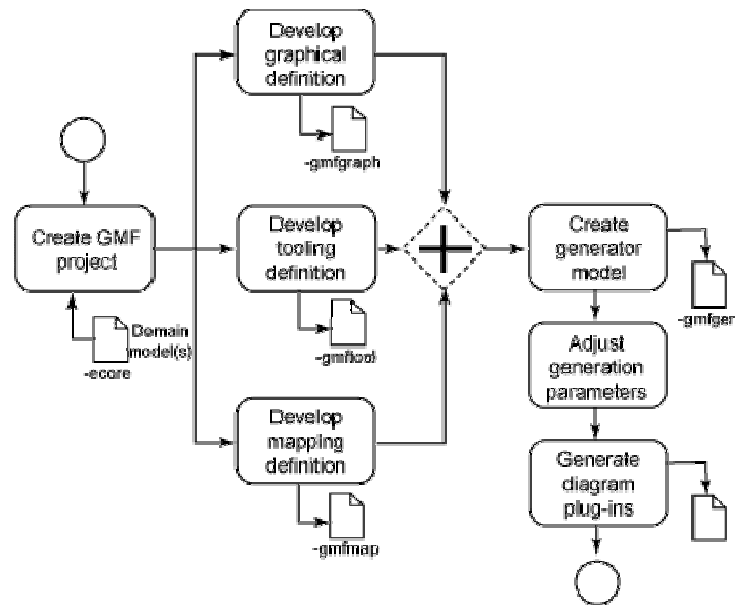
In its tooling definition TAJ includes tools for creating classes, attributes, operations, associations and specializations.

Mapping definition

This mapping is used by GMF to know the relations between the different definitions previously mentioned and the underlying ecore model. In this mapping we can specify which and how elements of the ecore model will be created and displayed by the editor.

The tooling definition is used by the mapping definition to associate an element in the editor palette to an element of the ecore model. At the same time the graphical definition is used to graphically display the different elements on the editor.

For example the mapping definition specifies that the compartments for attributes and operations defined in the graphical tool have to be drawn inside the Class element defined in the ecore model. Similarly here is were we define that the name attribute of a class object (defined in the ecore model) has to be displayed inside the representation of a class (specified in the graphical definition) using a label (also specified in the graphical definition), that the associations have to be displayed using a particular line style (previously defined in the graphical definition)...



GMF overview

It is important to understand the difference between the different elements involved in the creation of the editor. It is especially important to understand the difference between the ecore model and the graphical definition. Since the graphical editor stores information about the graphical model (eg: position of element, typographies, colors...), an element included in the graphical definition that does not have a mapping to the underlying model would be saved as part of the graphical information, while ignored by the model. Thus, a tool that uses the model as its input would not be able to access the information stored as part of the graphical definition.

At one point this problem appeared during the development process. At that point the model definition had to be recreated from scratch including the elements that were previously included as part of the graphical definition that were needed in the model. This forced us to recreate the subsequent elements implicated in the GMF creation too.

EMF

The EMF Eclipse project is a modeling framework and code generation facility for building applications based on a structured data model. Starting from a model specification EMF is able to generate a set of Java classes for the model. Those classes can be then used to generate new elements of the model specification. EMF also provides a based editor to manipulate the model.

GEF

The Graphical Editing Framework of Eclipse is the underlying component used by GMF to create the domain editor. GEF is an application neutral framework that provides the groundwork to build applications that need to make use of some graphical components.

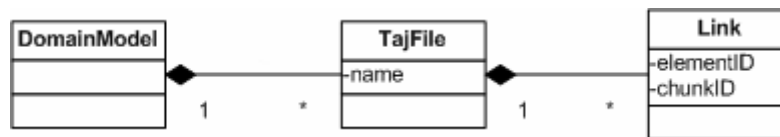
This framework is divided into 2 packages to support the needs for graphical editors like the document layout, rendering support, figures, connectors, printing, etc.

Link between code and domain

When the application domain has been created and the code has been annotated, the users can link those two sources of knowledge to reinforce their understanding.

The creation of those links implies the necessity to correctly identify the different elements. The chunks will be identified by a sequential number, while the domain elements are identified by a Uniform Resource Identifier (URI).

The domain model includes references to objects that store the information of the links created for each source code file with the domain model. Those links are merely a pair `<chunked, elementID>` that allow us to create the mapping “many to many” needed by our tool.



Linking structure

Conclusion

Program understanding happens at various levels of abstraction, but the different documentation artifacts are not clearly connected to source code constructions. By creating these two different elements we hope to achieve documentation that is constantly updated with the source code that can be easily browsed by the developer.

With the previous version of TAJ an experiment was conducted to test the effectiveness of the tool in relation with Javadoc or ordinary documentation. The results showed that TAJ performed better in both the lower level and the higher level of abstraction. The inclusion of the domain editor should only benefit this understanding. The obvious hypothesis would be that as the source code increases in size, the domain component becomes increasingly important to minimize the time needed to gain understanding of the application.

Eclipse is a very powerful platform to develop set of tools. Eclipse tries to create a low coupled environment where components can be added or removed to create a platform that can be adapted to different usages. At the same time, the core framework is surrounded by other components that try to ease the work of developer in different areas.

Unfortunately this creates a very steep learning curve; this is especially true when not only you need to learn the main platform, but also some of the surrounding technologies like EMF and GMF. Moreover the documentation available for the different component not always refers to the latest versions of the components, or when the documentation is simply not there. For instance the help available for GMF include a list of topics to learn the use of the platform, but some of those topics are not even written.

Future work

There are certain areas of TAJ that could be explored for further releases. These areas could range from user interface changes to create easier navigation between model and code, to some short of code generation to allow translation of source code to an object oriented design (independently of the design used in the previous code).

Editor capable of handling Java code

Currently, the TAJ editor is capable of creating chunks for a contiguous group of lines of code. Since sometimes more that one code construction is defined in a single line it would be useful to create chunks with smaller granularity, where chunk could contain only a portion of a given line.

Data storage method

In the actual version each java file annotated with TAJ generates a file that contains data generated by TAJ stored using serializable objects. The creation of a unique file that contains that information could reduce number of files to store and the disk space needed to store the data, thus improving maintainability.

The substitution of the serialized objects by a XML should also be studied, the use od the format would allow interoperability with other tools that could offer different views of the data created with TAJ.

Multiple domain definition

Current TAJ supports the creation of a single domain model. Since an application is affected by more than one domain, ranging from the main domain/s, to OS interaction, programming language knowledge, etc. it will be interesting to include the ability to create more than one domain for the same project.

The benefit of the ability to create more that one domain is not limited to the understanding created by the fact that more information is available for the user, but could also be used to eliminate undesired domains. Users could be allowed to define different views that would hide/show complete domains, change the text color in the source code so that users can easily identify code pertaining to different domains, etc.

Code generation/refactoring

New features could be added to the tool to allow refactoring of the source code by using some code generation technique. It would be possible to construct concrete classes for the abstract classes defined in the domain; these classes could be automatically filled with the different source code pertaining to chunks related to the classes. The code generated using this technique would create an object oriented structure even if the original source code was not creating this methodology.

This feature could also be used to translate between different programming languages.

Queries and statistics

Another interesting feature would be the ability to query the tool to obtain more complex information. For example user could request which code chunks are related to more than one domain, if the domains selected have little connection with each other but there are some chunks that are linked to both, those chunks could be candidates to code refactoring to reduce the coupling.

Better integration with Eclipse

Certain elements create using the Eclipse framework could benefit from a better integration with the platform. For example the outline view for TAJ documents could be used to quickly navigate through the code or to interact with the TAJ document structure (collapse, expand, change chunk description, etc.)

Papers

JRipples is a tool for during Incremental Change

J Buckner, J Buchta, M Petrenko, V Rajlich - Program Comprehension, 2005. IWPC 2005.

JRipples is a tool for assisting programmers during Incremental Change. JRipples automatically analyzes an application Java source files and extracts dependencies between classes. When a change has to be made to the application, the programmer has to select the initial class of the impact set, and JRipples is able to identify classes that might need revision based on the dependencies previously identified. Once those classes have been identified the tool assists the developer by keeping track of classes that have to be checked, classes that have been already modified by the developers to incorporate the new functionality and classes that do not need to be modified.

Similarly to TAJ, JRipples helps developers during the implementation of new functionality. The process followed by TAJ relies in a previously manually created annotated tree that can be used to easily identify which sections of code need to be modified to obtain a new version of the application with the new requirements implemented. By using per-line granularity programmers are able to identify more precisely sections of code that need revision. Another benefit of the TAJ application is that it can be used to reverse engineer an application source code and gain understanding about the application's high level architecture, this knowledge can be used as a reference when changes need to be made.

Design Fragments Make Using Frameworks Easier

G Fairbanks, D Garlan, W Scherlis - ACM SIGPLAN Notices, 2006

Given that programming frameworks impose certain burdens to programmers because the lifecycle of the application is affected by the framework, there is the need for programmers to comprehend how the framework and their code cooperate to complete a task. A way for programmers to learn how to use programming frameworks is to refer to examples, and copy code from those examples that provide the needed functionality. This paper proposes the creation of design fragments databases, these fragments include relevant information about how the framework has to be used to implement certain functionality in the application being built.

An IDE that support these design fragments can help developers checking for the correct implementation of the fragments, notify the users when a fragment has been deprecated or updated to solve certain bugs. Unfortunately the example provided in the paper is related to a relatively small framework like is Java Applets construction. From what I have experienced during this project I have learned that Eclipse framework is very complex, and even with examples and wizards, when you need to implement something slightly different, you find yourself “lost” in a sea of packages, interfaces, classes, and inheritance hierarchies. Designing and keeping updated fragments with new versions for all possible implementation needs seem an unaffordable task. At the same time it seems clear that there is a need for specific documentation that can be used to comprehend frameworks behavior.

Introduction and Overview Domain Analysis Concepts and Research Directions

G Arango, R Prieto-Diaz - Domain Analysis and Software Systems Modeling, 1991

This paper proposes domain analysis in the context of software reusability. The application of reusability techniques aims to help developers in the task of software development and maintenance. By making possible to reuse elements (modules, software architectures, formal transformations, test cases) software development complexity can be eased.

This paper highlights the difficulty of domain modeling in practice, where even small domains reveal themselves to be complex when they are under analysis.

Domains are an important part of the Eclipse plug-in we are developing. A domain is used to identify constructions in the code that are related to certain elements of the domain, in order to ease the process of locating those code constructions, there is the need to correctly identify the domain model used by the application at a appropriate granularity level. The difficulty of identifying a correct domain during the domain analysis must be taken into consideration when generation the domain used in the plug-in. Modifying the domain after finding certain code construction has to be done in such a way, that the element introduced in the domain model is not a simple translation from the source code, but a real element of the domain.

The use of domain knowledge in program understanding

S Rugaber - Annals of Software Engineering, 2000

The different documentation elements created during a software processes give usually a high level description of the solution to be implemented and the final requirements to fulfill. In contrast the source code is a very low level representation that is affected by many different elements like the programming language used, programming experience of the developers, non-functional requirements, etc. This gap between both representations can make it hard to relate the different elements between both representations. By understanding the different domains that affect the creation of the source code, developers know what to expect in terms of constructions in the source code, and how these constructions are related to elements in the domain.

Domains are an integral part of the TAJ tool. Software is created in order to solve a problem that is based in the real world, and thus those elements will affect somehow the source code. These relations, even if they are usually not explicitly represented, are needed in order to comprehend how the code achieves its results. By explicitly showing these relationships we believe that program understanding can be eased.

Improving the Quality of Requirements Specifications via Automatically Created Object-Oriented Models

D. Popescu, S. Rugaber, N. Medvidovic, D. M. Berry

Software requirements specifications (SRS) are a fundamental part of software construction. Even when the final version of the requirements can be written using in a formal language, the first draft is usually written in natural language (NL). The use of natural language can introduce ambiguities in the requirements specification, which can be hard to detect even after a review process.

The authors propose an approach to transform the requirements specification into a graphical model that can be reviewed more easily than the original requirement specification. Since the document has to be analyzed to create the diagram, the correct identification of elements used in the natural language is a key aspect for this approach to work. In order to accomplish this task, the authors limit the natural language used to a constraining grammar in order to reduce the complexity of the analysis procedure.

The paper examines how a NL SRS correctly identifies desired pieces of information when the NL is modified to the constraining grammar, but it fails to mention how much and how the text was altered in order to achieve the result shown. Also there is no study of how the review of the model performs against a review of the original NL SRS, given that the creation of the model could introduced errors that didn't exist in the original document, it would be interesting to compare results of these reviews.

SRS are used as a base to write the application code, and thus code is strongly related to the SRS. Since SRS are used to describe elements of a higher level domain it seems natural to use a SRS as a layer between of abstraction between the problem domain and the source code. TAJ is able to link source constructions with domain elements, the

ability of generating an initial model based on the analysis of the SRS could aid in the workflow of working with TAJ.

On the Knowledge Required to Understand a Program

R Clayton, S Rugaber, L Wills - Working Conference on Reverse Engineering, 1998

This paper reflects the complexity of understanding a program when the main source of knowledge of the program is the source code. Since applications are created to solve certain real life problem, the source code relates to elements in the domain of the problem, but the problem domain is not the only existing knowledge needed to create an application. When an application is created there is the need to know about the programming language used during the coding phase, general concepts about programming and how it relates to the underlying elements (hardware, virtual machine, interpreter, etc.). Those different knowledge domains affect the *ideal* design concept and add complexity when trying to move from the low level source code to a higher level of understanding.

The paper explores these issues with the source code of an application written in FORTRAN, which finds a root of a function of a single, real variable in an interval. The results of this study show that all the different domains that take part in the application source code are highly interleaved and equally important when trying to understand the application source code.

Allowing more than a single domain to be represented in TAJ is something that will have to be explored in future versions of the tool. This is not a trivial issue since certain code constructions can be, as shown in this paper, related to multiple domains and there is the need to keep all the different domains up to date in order for them to be useful.

Software Maintenance by Transformation

G. Arango, I. Baxter, .P Freeman, C. Pidgeon IEEE Software 3:33, 27-39, 1986

In this paper the authors explore the possibility of modifying existing operational software by transformation. They use a software model based in a tree structure where the root node is the initial system specification. Leaves are executable applications, and the intermediate nodes represent specifications at varying levels of abstraction. The arcs represent possible design choices. A specific software implementation of a specification is defined by a set of arcs that go from the root node to a leaf.

When a software product has to be modified, the model is traversed from the current leaf towards the root until a node that encompasses both the current and desired implementations is found, and then a new child node is explored until a new leaf node that implements all the needed requirements is reached.

Using this sort of techniques would be a very useful way of modifying existing software applications, but it seems to be based on an ideal model, where the different design decisions do not intersect with each other. For example a decision of changing certain software structure could invalidate certain states previously assumed as correct. This is something confirmed by the authors in their example, were they port an application from

UCI Lisp to Franz Lisp and they found the semantic gap between I/O concepts in the two dialects to be too large.

TAJ is based on the same concept that the final source code is an implementation of an ideal abstract representation (a domain), and that is affected by the different functional and non-functional requirements. The current version of TAJ is useful to gain knowledge of the application by abstracting software constructions to the domain concepts, but some of the concepts presented in the paper could be later added to TAJ in order to, for example, support a history of the evolution of the software, or the ability of generating portions of code automatically.

JIRiSS - an Eclipse plug-in for Source Code Exploration

D Poshyvanyk, A Marcus, Y Dong - Proc. of 14th IEEE International Conference on Program ..., 2006

JIRiSS is an eclipse plug-in for source code exploration, the plug-in allows users to search Java projects (source code and documentation) using natural language. The results of the queries are presented back to the user sorted by relevance, so that the most relevant results can be easily identified by the user.

Source code and documentation analysis and comprehension are a fundamental part of the process of software development, maintenance, reverse engineering, and software evolution, JIRiSS allows users to query those sources of information in order to extract knowledge from them. By integrating this process in the development IDE, users can more easily access the source of knowledge.

Unfortunately the paper does not include technical or implementation details, this fact makes it impossible for the reader to determine the usefulness of the plug-in compared to the search function included in all IDEs. In the paper they mention the ability to query the system using natural language, but the lack of information about the processing of the queries, makes it impossible to determine if there is some kind of semantic analysis of the queries or is just a simple use of keywords.

The ability to query a source of knowledge that includes both source code and documentation could be useful to determine associations between the low level implementation of the application and a higher representation of the problem that the application tries to solve. This could be beneficial in a project like TAJ, where the user has extracted knowledge from the code and link it to a domain for a better understanding of a program.

Supporting Document and Data Views of Source Code

ML Collard, JI Maletic, A Marcus - Proceedings of the 2002 ACM symposium on Document ..., 2002

This paper describes a representation of source code using the XML format. The XML stores the original source code and adds structural information to it, so various views can be selected from a single document. This XML document can be used by other applications to manipulate the document, and allows developers to query the document using XPath to extract information about it.

It is not clear how the use of this tool could be useful to developers since it seems to simply structural information that does not really add any real knowledge. Useful features like different highlight colors in the source code are available on any modern IDE and the creation of things like call graphs can be done directly from the source code. Also the authors affirm that srcML has higher visibility than AST & Symbol Table, but due to the tree-based structure of XML documents, there is limited expressiveness unless internal mapping are created between different elements of the document as a workaround. Since no document schema is shown is impossible to evaluate the expressiveness that can be archived with this representation.

For some reason they consider it important to maintain shallow information like whitespaces and tabulations, but it seems that it would be more useful to generate them when reading the document, offering the ability for developers to create they own formatting settings.

Also there is no actual implementation of the srcML tool so the authors could not show any tests results that can be used to evaluate their solution.

The use of an XML document to store the information generated by TAJ would be useful to allow interoperability with other tools. Since there are many tools and libraries to manipulate XML documents this addition would make possible to offer different views of the same information. This could lead to views for certain users that hide non-relevant data, so the relevant information is more visible to them.

References

- Eclipse 3.0 Porting Guide, Generic Workbench

http://dev.eclipse.org/viewcvs/index.cgi/platform-ui-home/rcp/generic_workbench_porting_guide.html?view=co

- Folding in Eclipse Text Editors

<http://www.eclipse.org/articles/Article-Folding-in-Eclipse-Text-Editors/folding.html>

- OSGi

<http://www.osgi.org/>

- The Official Eclipse FAQs

http://wiki.eclipse.org/index.php/Eclipse_FAQs

- GMF Tutorial

http://wiki.eclipse.org/index.php/GMF_Tutorial

- GMF New and Noteworthy

http://wiki.eclipse.org/index.php/GMF_New_and_Noteworthy

- Generating an EMF Model

- The Eclipse Modeling Framework (EMF) Overview