View Infinity

Zoombare Ansichten für die Arbeit mit Feature-Diagrammen im Bereich Software-Visualisierung

Institut für Simulation und Graphik
Fakultät für Informatik
Otto-von-Guericke-Universität

Projektorganisation: Jun.-Prof. Dr.Ing. Raimund Dachselt
dachselt@acm.org

Entwicklung: Michael Stengel
virtuellerealitaet@googlemail.com

Unterstützung: Mathias Frisch, Christian Kästner, Janet Feigenspan, Marco Winter

Postanschrift:
Jun. Prof. Dr.-Ing. Raimund Dachselt
Otto-von-Guericke-Universität
Institut für Simulation und Graphik
Fakultät für Informatik
39106 Magdeburg
Germany

Inhalt

1 Einleitung

In der heutigen Zeit können sich Anwender bereits in vielen Bereichen ihre Produkte selbst konfigurieren, sei es bei der Auswahl der zu installierenden Komponenten einer Software oder die Farben des neuen Autos. Im Bereich der Softwareentwicklung werden Auswahlmöglichkeiten der Komponenten einer Anwendung als „Features“ bezeichnet. Im Quellcode können Features über Parameter gesteuert werden, die vom Präprozessor des Compilers beim Erstellen der Anwendung ausgewertet werden. Für Softwareentwickler ist es allein anhand des Quelltextes schwierig zu erkennen, welcher Teil des Quellcodes für ein bestimmtes Features relevant ist. Dies ist insbesondere dann der Fall, wenn der Entwickler mit der Software noch nicht vertraut ist. Der Code für Features kann sich überall im Quellcode verteilen, wodurch die Suche im Quelltext aufwendig und zeitintensiv ist. Um die Arbeit mit Features in Softwareprojekten zu vereinfachen, wurde im Rahmen der Veranstaltung „Software & Model Visualization“ ein Programm entwickelt, welches eine effiziente Interaktion mit dem Feature-Modell erlaubt. Die Anwendung ermöglicht eine interaktive Darstellung des Feature-Graphen, der dazugehörigen Dateistruktur des Projektes, sowie des relevanten Quelltextes. Das Projekt dient vorrangig der Projektverwaltung und richtet sich demnach an Projektmanager und Neueinsteiger für die Projektentwicklung.

2 Programminstallation

View Infinity wurde unter Microsoft Windows 7 (x64) und Microsoft Visual Studio C++ 2008 Express entwickelt. Das Kompilieren sollte jedoch auch unter anderen Windows-Versionen (auch X86) funktionieren, da die verwendete Bibliotheken Open Graph Drawing Framework (OGDF) auf beiden Architekturen funktioniert und das verwendete Toolkit Qt ebenso auf 32-Bit-Systemen läuft.

Ausführen der vorkompilierten Anwendung

Für die Installation muss zunächst das Archiv **`ViewInfinity.zip`** entpackt werden. Das Archiv enthält bereits unter Windows 7 (x64) und Visual Studio 2008 kompilierte Binaries.

Um die Anwendung auf einem solchen System zu benutzen, reicht in der Regel das Starten der **`ViewInfinity.exe`** im Ordner **`Release`** im entpackten Ordner aus.

Dem Projekt liegen für die Benutzung der Anwendung zwei Beispielprojekte im relativen Pfad **`Examples`** bei (**`ExpressionP`**, **`Prevayler`**).

Neukompilieren der Anwendung

Schritt 1: Kompilieren der COIN-Bibliothek und des Open Graph Drawing Frameworks

Für die Benutzung der Anwendung auf einem anderen System ist es nötig, die Anwendung, sowie die Bibliothek OGDF noch einmal darauf zu kompilieren. Zu diesem Zweck liegen die Quellcodedaten sowie die Visual Studio Projektdaten der Anwendung in der Datei **`ogdf_library_src_bin_vs2008.zip`** bereit. Der Inhalt dieser Datei sollte in das Root-Verzeichnis von View Infinity entpackt werden, um die Verzeichnisstruktur zu wahren.

Anschließend kann das Projekt **ogdf.vcproj** kompiliert werden. Die OGDF-Bibliothek sollte somit im Root-Ordner den relativen Dateipfad **lib/OGDF.lib** (in der Release-Version) bzw. **lib/OGDFd.lib** (in der Debug-Version) besitzen.

Für das Kompilieren von OGDF wird die COIN-Bibliothek (Version 1.3.1) benötigt. Die Binaries der COIN-Bibliothek für Visual Studio 2008 sind bereits im mitgelieferten Archiv der OGDF-Bibliothek enthalten. Treten bei der Erstellung der OGDF-Bibliothek Probleme auf, muss möglicherweise auch die COIN-Bibliothek neu erstellt werden. Der nötige Quellcode und das Visual Studio Projekt ist im Archiv **COIN1.3.1_src.zip** vorhanden und muss ebenfalls in das Root-Verzeichnis von View Infinity entpackt werden. In der Projektmappe **CoinAll.sln** müssen die Projekte **clp**, **libCoinUtils**, **libOsi** und **libOsiClp** erstellt werden. Im Anschluss daran sollte OGDF kompiliert werden können.

COIN 1.3.1 kann ebenso von der Entwicklerseite heruntergeladen werden.
http://www.coin-or.org/download/source/CoinAll/

Schritt 2: Installation des Qt Toolkits

View Infinity wurde zusammen mit Qt 4.6.0 entwickelt. Möglicherweise funktioniert die Anwendung auch mit neueren Versionen von Qt. Die Binaries von Qt für Windows können von der Entwicklerseite heruntergeladen werden. http://qt.nokia.com/downloads/ Qt wird anschließend über das Setup installiert. Für das Verwenden von Qt in Visual Studio muss eine Systemvariable **QTDIR** mit dem Pfad angelegt werden, in dem sich die Binaries von Qt befinden (z.B. **C:\Qt\4.6.0**). Für die das spätere Ausführen von View Infinity müssen die Qt-Bibliothken **QtCore4.dll**, **QtGui4.dll** und **QtXml4.dll** aus dem **Bin**-Verzeichnis von Qt in das Verzeichnis **Release** (bzw. **Debug**) von View Infinity kopiert werden. Vorhandene Dateien sind zu überschreiben.

Schritt 3: Erstellen der View Infinity Anwendung

Liegen die Qt-Bibliotheken und die OGDF-Bibliothek in den beschriebenen Verzeichnissen von View Infinity, kann das Projekt **View Infinity** in der gleichnamigen Projektmappe **View Infinity.sln** unter Visual Studio erstellt werden. Im Anschluss daran liegt die ausführbare Datei **ViewInfinity.exe** im **Release**-Ordner (bzw. im **Debug**-Ordner) vor und kann ausgeführt werden.

2 Visualisierungskonzepte

Für die effektive Visualisierung von Quellcode gibt es verschiedenste Visualisierungskonzepte die ausführlich in der Vorlesung „Software & Model Visualization“ bei Prof. Raimund Dachselt behandelt wurden. Die Konzepte die für die Umsetzung des Projekts in Betracht kamen, sowie einleitend der generelle Aufbau, werden im Folgenden erläutert.

2.1 Genereller Aufbau

Das Konzept der Anwendung basiert auf vier verschiedenen Ansichten des betrachteten Softwareprojektes und umfasst unterschiedliche Abstraktionsgrade. Dies ist zum Einen das Feature-Modell als Baumhierarchie in einem Fenster, des Weiteren die Ansicht der

Dateistruktur der Projektdaten, die Ansicht des zugehörigen Quellcodes und abschließend das entsprechende UML-Diagramm.

2.2 Graphen und Bäume

Die drei gegebenen Test-Projekte sind feature-orientiert programmierte Anwendungen und enthalten daher Daten über den generellen Projekt-Aufbau sowie eine „Model“-Datei, in welcher die Features und deren Abhängigkeiten beschrieben sind. Die Idee dieses Projektes ist, diese Dateien in die vier genannten Ansichten zu überführen und interaktiv miteinander zu verbinden. Als Datenstruktur kommt für das Feature-Modell und die Dateistruktur jeweils ein Graph, genauer ein Hierarchiebaum, zum Einsatz. Für das Layout von Bäumen gibt es vielerlei Möglichkeiten wie beispielsweise das Reingold-Tilford-Layout, Balloon Trees, Tree Maps oder auch 3D-Layouts, wobei für das interaktive Feature-Diagramm für beide Bäume eine vorrangig eine hierarchische Baumvisualisierung nach Reingold und Tilford in 2D vorgesehen ist.

2.3 Hilfsmittel zur Visualisierung von Features

Für die Arbeit mit Quellcode als textuelle Darstellungen kann mittels Farbe, Schriftgröße oder Schriftart ein Fokus erzeugt werden, um dem Benutzer das Auffinden wichtiger Bereiche im Dokument zu erleichtern. In diesem Projekt soll für jedes Feature, welches im Graphen selektiert wird, der entsprechende Quellcode der Datei angezeigt werden und die zum Feature gehörenden Quellcodeabschnitte farbig hervorgehoben werden. Dadurch soll die Suche nach relevanten Codezeilen erleichtert werden. Da Softwareentwickler mit der farbigen Hervorhebung von Standard-Statements, Suchergebnissen oder auch Syntaxfehlern im Quellcode vertraut sind, ist die Verwendung einer farbigen Textpräsentation naheliegend. Bei der Auswahl von mehreren Features können mehrere, aber eindeutige Farben verwendet werden um die zugehörigen Textabschnitte den Features zuordnen zu können. Enthält ein Abschnitt allerdings mehrere Features würde Farbe allein für die Darstellung aufgrund der Mehrdeutigkeit nicht ausreichen. Zur Lösung der Problematik kann beispielsweise mit umschweifenden Klammern, Mehrfachdarstellungen, weiterer Filterung oder einer texturierten Visualisierung die Assoziation zu den entsprechenden Features eindeutig hergestellt werden. Für die Visualisierung von Features und Dateien in den zugehörigen Graphen bzw. deren gegenseitigen Abhängigkeiten eignen sich zum Beispiel die Parameter Farbe, Position, Größe, Verbindungslinien, aber auch Symbole oder Icons. Für das Projekt „Interactive Feature Model“ werden diese Hilfsmittel auch genutzt, um Zugehörigkeiten und auch den Status (aktiviert / deaktiviert) von Features, Klassen und Quellcode-Dateien in einem Software-Projekt zu verdeutlichen.

2. 4 Interaktionsmöglichkeiten

Wie bereits im Namen des Projekts manifestiert, liegt der Schwerpunkt dieser Anwendung auf den Interaktionsmöglichkeiten. Die effiziente Navigation und Interaktion im Graphen und mit den Quellcodedaten ist insbesondere bei großen Softwareprojekten (> 10.000 „Lines Of Code“) wichtig, da Datei-Graphen oder Klassendiagramme mit vielen Knoten unübersichtlich oder nicht mehr komplett darstellbar sind. Daher sind Interaktionsmöglichkeiten zur dynamischen Anpassung der Ansicht unabdingbar. Dazu gehören beispielsweise „Zooming“, „Panning“, „Folding / Unfolding“, „Brushing und Linking“, Selektion, Überblicks- und Detailansicht, die Wahl unterschiedlicher Layouts oder auch das „Neu-Arrangieren“ der Ansicht. Die Zoom-Funktion kann dabei noch in geometrisches und semantisches Zoomen unterschieden werden. Für dieses Projekt wurden Zooming und Panning als grundlegende

Interaktionsmöglichkeiten festgelegt, da sonst die Beschriftungen und auch Abhängigkeiten in den Graphen nicht mehr erkennbar wären, sobald Projekte mit sehr vielen Features geladen werden. Um im Dateiansichtsgraphen einen möglichst guten Überblick zu erhalten, soll hier ein Folding / Unfolding möglich sein um bestimmte Subgraphen ein- und ausklappen zu können. Features sollen aktiviert und deaktiviert werden können (Selektion). In Abhängigkeit des selektierten Features soll sich die Code- und UML-Ansicht entsprechend anpassen. Das durchgängige Konzept eines Zoomable Interfaces erlaubt das intuitive „Durchfahren“ aller verfügbaren Ansichten des Projektes. Unterschiedliche Levels-Of-Detail des dargestellten Datei-Graphen ermöglicht dem Benutzer das Behalten der Übersicht in großen Projektstrukturen. Beim Zoomen wird das Level-Of-Detail dynamisch dem Zooming-Level angepasst, wodurch die angezeigte Darstellungsqualität optimiert wird. So sind bspw. Datei-Knoten in der niedrigsten Detailstufe als einfarbige Knoten dargestellt, die durch ein festgelegtes Farbschema eine präattentive Wahrnehmung der Knotentypen zulässt. Bei höheren Detailstufen werden dynamisch Knoten-Symbole, Beschriftungen und Code-Fragment-Visualisierungen eingeblendet.

3 Implementierung

3.1 Genereller Aufbau

Entwickelt wird das Projekt in der Sprache C++ in Microsoft Visual Studio 2008, wobei als Grundlage für die Implementierung das Qt Graphis Toolkit in der Version 4.6.0 und die Graphenbibliothek OGDF (Open Graph Drawing Framework) in der Version v.2007.11 (Bubinga) genutzt wird. Für die Umsetzung des Graphical User Interfaces (GUI) und das Zeichnen der graphischen Elemente kommt Qt zum Einsatz. OGDF wird genutzt, um für die Projektdaten nötige Graphenstrukturen aufbauen und nutzen zu können.
Wie bereits im Konzept beschrieben, können vier verschiedene Ansichten des Projektes (Feature-Modell, Dateistruktur, Quellcode) über das Zoomable Interface durchlaufen werden. Die einzelnen Ansichten, wie auch die Klassenansicht, sind außerdem direkt über Schaltflächen im Hauptfenster erreichbar. Um die Navigation in der Datei-Ansicht des Projektes zu vereinfachen, ist im linken Bereich des Programms zusätzlich die Hierarchie der Dateistruktur über interaktive Einrückungen visualisiert („Tree View“ oder auch „Indentation View“).

3.2 Graphen und Bäume

Die Visualisierung der Projektstruktur erfolgt über die Darstellung von hierarchischen Bäumen, eine für das Feature-Modell und zwei für die Darstellung der Projekt-Dateien. Dabei wurden das Feature-Model aus der „model.m“- Datei ausgelesen und der Dateienbaum aus der „annotations.xml“. Anschließend erfolgt der Aufbau eines Feature-Graphen und eines Dateigraphen im OGDF-Format. Für diese Graphen wird voreingestellt über den Sugiyama-Algorithmus ein Layout berechnet und diese in der Anwendung dargestellt. Weitere Layouts sind über ein Menü aufrufbar.

3.3 Implementierungsansichten

3.3.1 Feature Modell

In der Feature-Modell-Ansicht wird eine interaktive Übersicht über die vorhandenen Feature und deren Abhängigkeiten zueinander als Baum visualisiert. In dieser Ansicht können Features aktiviert und deaktiviert werden, wobei eine Verlinkung zur Datei- und Code-

Ansicht erfolgt. Die Visualisierung der Feature-Abhängigkeiten erfolgt über die folgende Notation:

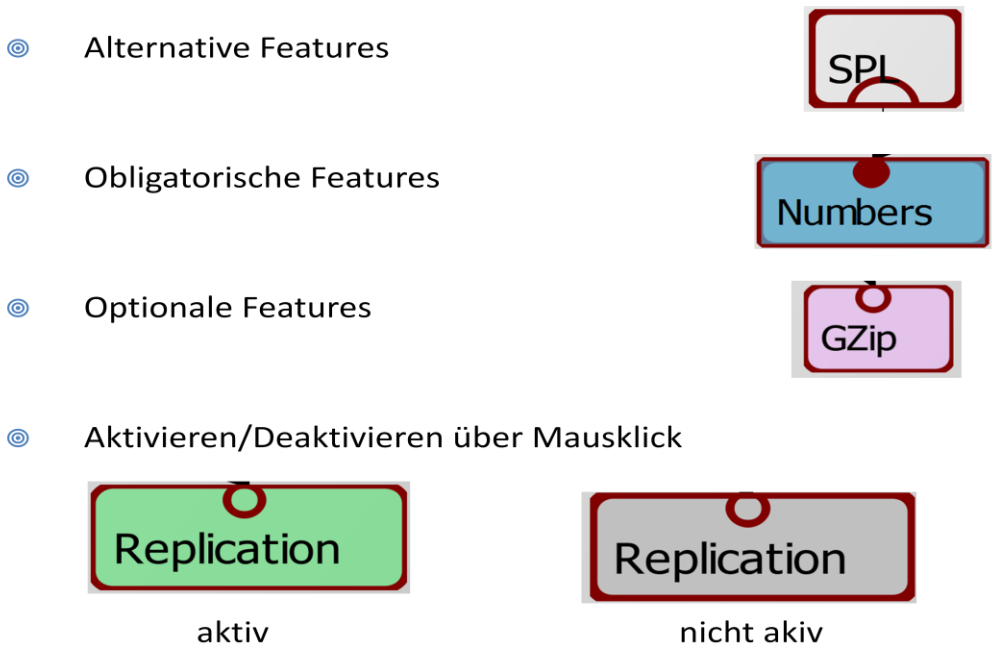

Abb 1: Notation der Features

3.3.2 Datei-Struktur

Die Datei-Struktur-Ansicht erfolgt ebenfalls als Baum und visualisiert den Aufbau der Projektdateien. Dateien die zu Features, die in der Feature-Modell-Ansicht nicht ausgewählt wurden, werden gelöscht. Es erfolgt des Weiteren eine Synchronisation zwischen der Ansicht des Datei-Graphen und der „Tree View"- Datei-Ansicht links. Hier ist es möglich Ordner der Projektstruktur einzuklappen und dies in beiden Ansichten zu visualisieren.
In der Datei-Graphen-Ansicht ist das semantische Zooming hervorzuheben. Dadurch werden verschiedene Levels-Of-Detail angezeigt. Dies ist zum Einen eine Fragment-Ansicht. Um die Verteilung der Features in einer Datei zu veranschaulichen, werden feature-spezifische Farben Rechtecken zu gewiesen, die auf den Dateiknoten gemappt werden. Dadurch erhält man einen guten Überblick über die Anzahl und Verteilung der einzelnen Features in einer Datei. Das nächste LOD ist eine verkleinerte Version des Datei-Graphen, bei der die textuelle Beschriftung entfernt wurde. Das letzte LOD ist eine weiter vereinfachte Knotendarstellung. Hier werden die Dateiknoten komplett eingefärbt, je nachdem, ob es sich um einen Ordner, einen Projektknoten oder eine Datei handelt.

3.3.3 Quellcode

Die Codeansicht einer Datei wird dargestellt, sobald der Nutzer an eine bestimmte Datei in der Datei-Ansicht nahe genug heranzoomt. Die Darstellung des Quelltextes ist mit einem Syntaxhighlighting versehen, welches die Standardausdrücke wie Methoden, Bedingungen Klassendefinitionen farbig hervorhebt. Der zu dem jeweiligen Feature gehörende Text wird

in der jeweiligen Feature-Farbe eindeutig eingefärbt. Über ein „Tool Tip“ sind weitere Informationen über das Feature verfügbar.

3.3.4 Klassendiagramm

In der Klassendiagramm-Ansicht soll das UML-Diagramm zu der Datei, welche zum gewählten Feature zugehörig ist, angezeigt werden. Diese Ansicht ist in der derzeitigen Version noch nicht integriert.

3.4 Visualisierunghilfsmittel

Features im Feature-Graphen werden zur visuellen Unterstützung farblich hervorgehoben. Ein aktives Feature erhält einen in der Feature-Farbe farbigen, leicht transparenten, Hintergrund, wobei ein inaktiver Knoten einen grauen Hintergrund erhält und Nicht-Features einen 100%ig transparenten Hintergrund. Auch für die Visualisierung der Abhängigkeiten wurde sich unterschiedlicher Farben und Formen, wie in der Notation beschrieben, bedient. Optionale Abhängigkeiten werden durch einen roten, nicht-gefüllten Kreis dargestellt, obligatorische Abhängigkeiten dagegen durch einen ausgefüllten roten Kreis am oberen Rand eines Knotens, Alternativen werden durch einen leeren Halbkreis und OR-Verbindungen durch einen ausgefüllten Halbkreis visualisiert. Im Dateiengraphen kommen Icons abhängig vom Typ eines Knotens zur Anwendung.

3.5 Interaktionsmöglichkeiten

Die geplanten Interaktionsmöglichkeiten konnten umgesetzt werden. So können Features im Graphen aktiviert und deaktiviert werden und in der Codeansicht kann der zum selektierten Feature zugehörige Code angezeigt werden. Die Knoten können im Feature-Graphen und im Datei-Graphen beliebig verschoben werden und ein geometrisches Zooming ist für alle Ansichten implementiert. Im Datei-Graphen können Subgraphen durch Klicken auf das Folding/Unfolding-Symbol eines Knotens, eingeklappt oder ausgeklappt werden und diese Interaktion ist verlinkt mit der zweiten Dateiansicht in der linken Seite, auch dort werden die Knoten ein- bzw. ausgeklappt und umgekehrt. Das semantische Zooming ist in der Datei-Strukturansicht verfügbar und kann wahlweise über das Rad der Mouse oder die „Zoom“ – Schaltflächen im linken Fenster erreicht werden.

3.6 Anwendung

Für die Arbeit mit dem Feature-Modell eines Java-Projektes ist das Vorhandensein der zugehörigen "model.m", "annotations.xml", sowie die Quellcode-Dateien des Projektes nötig (siehe http://wwwiti.cs.uni-magdeburg.de/iti_ db/research/cide/xml).

Geöffnet werden kann ein Projekt über das Menü (File->Open), dabei muss ein Ordner ausgewählt werden, der sowohl die „model.m“ - Datei als auch die „annotations.xml“- Datei enthält. Die Graphen werden dann automatisch erstellt und sind sofort verfügbar. Weiterhin kann über die Schaltflächen oben links zwischen den Ansichten gewechselt und gezoomt werden. Diese werden dann im Hauptfenster dargestellt werden. Zooming kann entweder innerhalb des Hauptfensters über das Rad der Mouse erfolgen oder auch über die „Zooming“-Buttons.
Innerhalb der Graphenansichten kann über "Drag and Drop" mit der linken Maustaste jeder Knoten verschoben werden. Mit einem Mausklick können Features aktiviert oder deaktiviert werden und auch Subgraphen ein- und ausgeklappt werden. Auch eine Exportmöglichkeit des Graphen in der "Graph Markup Language" (GML) steht zur Verfügung.

3.7 Einschränkungen

Für kleinere Projekte ist die Anwendung gut zu handhaben, bei größeren Projekten werden aber die Nachteile einer Baumansicht bemerkbar, beispielsweise, dass Knotenbeschriftungen nur bei höheren Zoomstufen lesbar sind und auch die Abhängigkeitsmarkierungen unter Umständen nur nach einem Reinzoomen erkennbar sind. Auch wäre für die Notation der Abhängigkeiten und den Status eines Knotens in der Feature-Modell Ansicht eine dauerhaft eingeblendete Legende hilfreich. Ein Problem stellt auch die Performance bei größeren Projekten dar, da das Layout des Graphen bei jedem Folding/Unfolding und Ändern von Features komplett neu berechnet werden muss. Des Weiteren führen hohe Auflösungen zu niedrigen Bildraten.

4 Ausblick

Zukünftig soll noch die beschriebene Klassendiagramm-Ansicht integriert werden und das semantische Zoomen verbessert werden. Des Weiteren könnten die Abhängigkeitsdarstellungen in der Feature-Modell Ansicht verändert werden Bis jetzt orientieren sich die Notationen an den bereits vorhandenen aus den Test-Projekten, und sind wie bereits erwähnt ohne Legende nur über die Tool Tips nachvollziehbar. Diese Aspekte geben Anlass zur Weiterentwicklung des vorhandenen Prototyps.

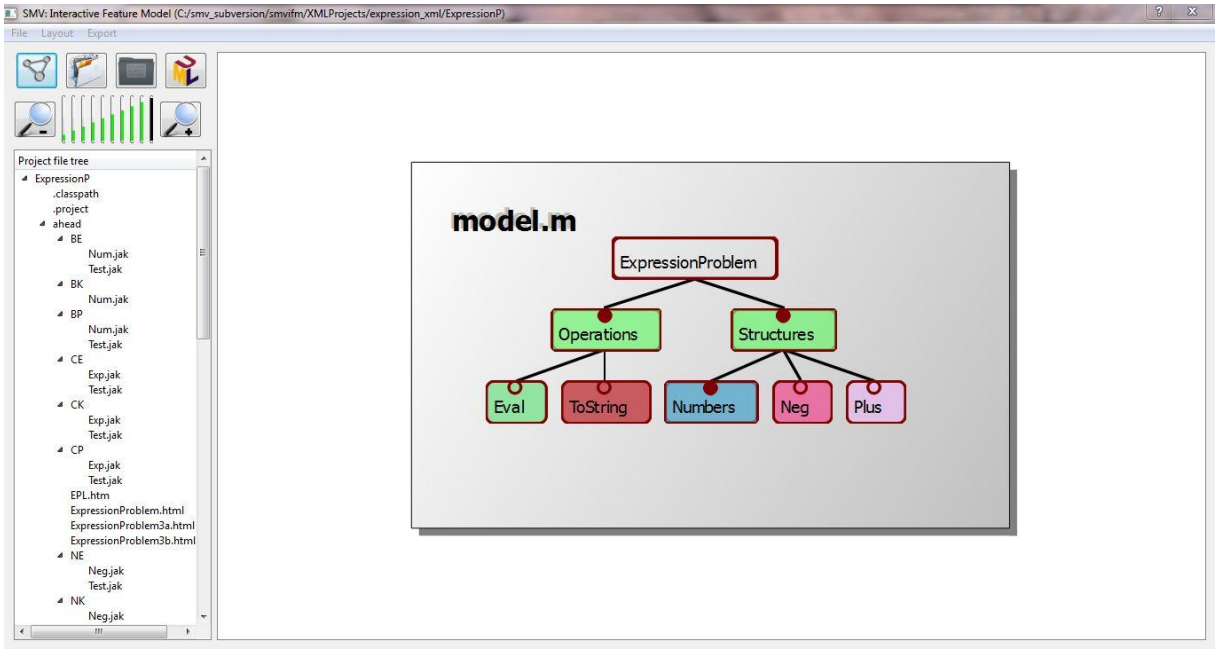

Abb 2: GUI der Anwendung