DELTA software technology

NEWSLETTER

AUSGABE MÄRZ 2013

IN DIESER AUSGABE

Kontaktieren Sie uns

Delta Software Technology GmbH
Eichenweg 16
57392 Schmallenberg

phone +49 2972 9719-0
fax +49 2972 9719-60
e-mail info@delta-software.com

www.delta-software.com

1 Frühjahrsputz für COBOL- und PL/I-Anwendungen

Der Frühling naht und die Sonne bringt Licht in Wohnungen und Häuser. Staub und Schmutz, die sich in der dunklen Jahreszeit ungesehen ansammeln konnten, werden sichtbar, so dass in vielen Haushalten nun ein Frühjahrsputz beginnt. Warum den Frühjahrsputz nicht auch gleich auf Ihre COBOL- und PL/I-Anwendungen ausweiten? Solche Anwendungen sind über viele Jahre oder sogar Jahrzehnte gewachsen. In dieser Zeit haben sie gelebt - sie wurden gewartet, erweitert, an neue Anforderungen angepasst und sind dadurch gewachsen. An vielen Stellen hat sich überflüssiger Ballast angesammelt, Programmstrukturen haben sich geändert, Dokumentation ist nicht mehr vorhanden oder nicht mehr aktuell. Die Wartung und das Verstehen der Anwendung werden immer schwieriger.

AMELIO Logic Discovery extrahiert die implementierte Anwendungslogik aus COBOL- und PL/I-Programmen und hilft somit die Anwendungen zu verstehen.

Die Ermittlung der Anwendungslogik erfolgt in den drei Schritten Inventur, Code Optimierung und Logikanalyse. Die Inventur ermittelt, aus welchen Bestandteilen – also Programmen, Schnittstellen und Datenstrukturen, sowie den entsprechenden Zusammenhängen - sich die Anwendung zusammensetzt. Bei der Code-Optimierung werden die für die Wartung und das Verstehen relevanten Anteile der Anwendung ermittelt, dabei wird z.B. toter Code festgestellt, dokumentiert und ggf. entfernt. In der Logikanalyse werden aus den Informationen der vorangegangenen Schritte Modelle erzeugt, die sowohl unabhängig von den verwendeten Programmiersprachen als auch den -paradigmen sind. Auf diese Weise wird die Anwendungslogik extrahiert und verständlich repräsentiert.

Die Inventur sowie die Dead Code -Analyse und -Bereinigung aus der Code Optimierung haben wir im CleanUp-Paket von AMELIO Logic Discovery zusammengefasst.

Dieses Paket ist dazu gedacht, existierende Anwendung zu redokumentieren und vor allem unnötigen Ballast zu erkennen, zu

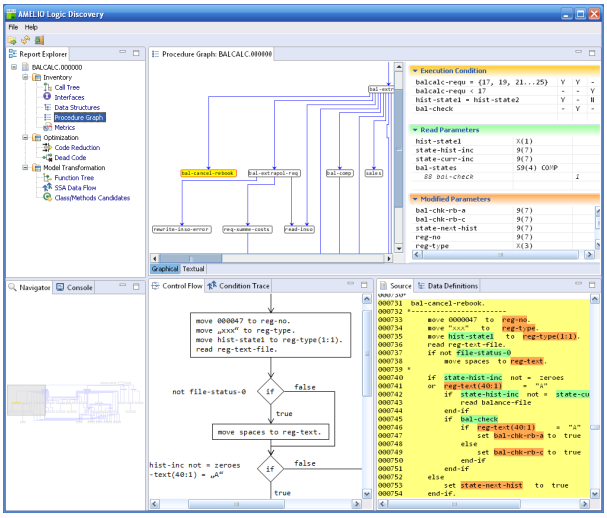
dokumentieren und zu entfernen und somit effizient, zuverlässig und automatisch den Frühjahrsputz durchzuführen.

Inventur – Mehr als nur eine Bestandsaufnahme

Der erste Schritt, sowohl beim Frühjahrsputz als auch beim Verstehen einer Anwendung, besteht darin festzustellen, was sich in der vergangenen Zeit alles angesammelt hat. In der Inventur wird deshalb z.B. ermittelt, aus welchen Programmen sich die Anwendung zusammensetzt, welche Schnittstellen und Datenstrukturen es gibt und welche Zusammenhänge bestehen. Hierbei geht AMELIO Logic Discovery über eine reine Bestandsaufnahme hinaus, stattdessen werden bereits Analysen durchgeführt:

- Klassische Metriken, z.B. Halstead oder McCabe
- Auflistung existierender Datendefinitionen
- Auflistung aller Datenbank- und Dateizugriffe
- Auflistung aller Schnittstellendefinitionen und möglicher Konflikte
- Darstellung von Programm- und Unterprogrammaufrufen
- Darstellung von Prozeduren, ihren Aufrufhierarchien, Bedingungen und Schnittstellen

In der Analyse-Workbench von AMELIO Logic Discovery werden die Ergebnisse der verschiedenen Analysen dargestellt. Als Beispiel für eine solche Analyse und der Darstellung ihrer Ergebnisse wird im Folgenden ein besonderes Highlight der Inventur vorgestellt: Die Repräsentation von Prozeduren zusammen mit ihrer Aufrufhierarchie, den Schnittstellen, verwendeten Parametern und vor allem den Bedingungen, wann die jeweilige Prozedur ausgeführt wird. Da es in COBOL keine expliziten Prozeduren gibt, wird jedes Programm der Anwendung analysiert. Anhand der vorgefundenen Aufrufstrukturen werden dann verschiedene Abschnitte des Programms zu Prozeduren zusammengefasst. Für PL/I, wo es explizite Prozeduren gibt, können diese als Ausgangsbasis für die folgenden Analysen verwendet werden, allerdings verbergen sich hier viele Prozeduraufrufe hinter Entry-Variablen und Generics, die erst aufgelöst werden müssen. Die gefundenen Prozeduren und ihre Aufrufstruktur werden in Form eines Graphen dargestellt. Das Analyse-Ergebnis ist beispielhaft in der folgenden Grafik dargestellt:



- Prozedurgraph: Stellt die ermittelten Prozeduren und ihre Aufrufhierarchie dar, ebenso die Aufrufe von Unterprogrammen und „wilde“ Verzweigungen mit GOTO.
- Bedingungen: Im Rahmen der Inventur erfolgt bereits eine Bedingungsanalyse. Bei dieser Analyse wird ermittelt, welche Bedingungen erfüllt sein müssen, damit eine Prozedur überhaupt

ausgeführt wird. Das Ergebnis wird in Form von Entscheidungstabellen dargestellt.

- Datenstrukturen (COBOL): Es wird dargestellt, welche Datenstrukturen durch die Prozedur gelesen und welche modifiziert werden.
- Datenstrukturen (PL/I): Es wird dargestellt, welche globalen Datenstrukturen durch die Prozedur gelesen und welche modifiziert werden.
- Schnittstellen: Speziell für PL/I-Anwendungen wird zusätzlich die Schnittstelle einer Prozedur angegeben und falls vorhanden auch deren Rückgabewert.
- Kontrollfluss: Zu jeder Prozedur wird deren Kontrollfluss angezeigt.
- Code-Anteile: Darstellung, welche Codeanteile die Prozedur bilden, darin werden gelesene und modifizierte Datenstrukturen farblich markiert.

Toter Code – Unnötigen Ballast entfernen

Je länger eine Anwendung lebt, desto öfter wurde sie erweitert und angepasst. Von daher ist davon auszugehen, dass mit dem Alter der Anwendung die Menge toten Codes in der Anwendung steigt. Also Code, der zwar vorhanden ist, aber nie ausgeführt werden kann. Solcher Code muss bei Wartungsmaßnahmen immer mitberücksichtigt werden, er erschwert das Verstehen der Anwendungslogik und belegt Speicherplatz. Neben der Inventur ist deshalb die Erkennung und Beseitigung des toten Codes ein zentraler Bestandteil des Frühjahrsputzes.

Die Dead Code-Analyse wird pro Programm durchgeführt, sie setzt sich aus den folgenden Bestandteilen zusammen, die aufeinander aufbauen:

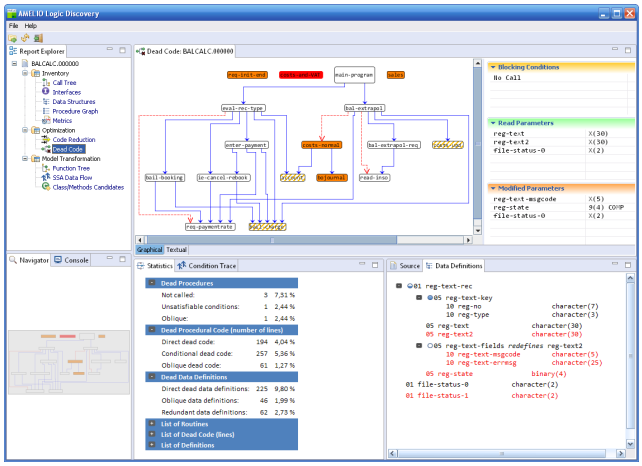
- Direct Dead Code: prozeduraler Code, der nicht ausgeführt wird, da der entsprechende Abschnitt nie aufgerufen wird
- Conditional Dead Code: prozeduraler Code, der nur unter bestimmten Bedingungen ausgeführt wird (z.B. innerhalb einer If-Anweisung), die entsprechende Bedingung jedoch nie erfüllt werden kann
- Oblique Dead Code: Dies sind prozedurale Code-Anteile, die nur aus dem zuvor gefundenen toten Code aufgerufen werden
- Direct Dead Data Definitions: Datenstrukturen, die definiert wurden, aber nie verwendet werden
- Oblique Dead Data Definitions: Datenstrukturen, die definiert wurden, aber nur innerhalb von totem prozeduralem Code verwendet werden
- Redundant Data Definitions: Stellen in dem Sinne keine Dead Data Definitions dar, sondern sind durch kopieren und einfügen entstanden, dennoch können sie im Rahmen der Dead Code-Analyse ermittelt werden.

Das Ergebnis der Dead Code-Analyse wird in der folgenden Graphik dargestellt:

- Reduzierter Prozedurgraph: Dieser Graph enthält alle toten Prozeduren (in orange eingefärbt, die selektierte Prozedur ist rot eingefärbt) und alle Prozeduren, die toten Code enthalten (orange schraffiert). Wird eine Prozedur als tot erkannt, weil die Bedingung zu ihrer Ausführung nie erfüllt werden kann, so ist der entsprechende Aufruf orange markiert. Außerdem werden alle Prozeduren angezeigt, die entweder in der Aufrufhierarchie oberhalb einer der zuvor genannten

Prozeduren liegen oder alternative Aufrufpfade zu einer Prozedur darstellen, die aus einer toten Prozedur aufgerufen werden.

- Blocking Condition: Die Bedingungen, die eine Ausführung der Routine verhindern.
- Data Definitions: Stellt die Datendefinitionen dar. Tote Datendefinitionen sind rot markiert.
- Statistics: Statistik für das gesamte Programm. Sie gibt an wie viele tote Routinen, Zeilen prozeduraler Code und Datendefinitionen ermittelt wurden. Außerdem werden die entsprechenden Elemente aufgelistet.



Erweiterte Dead Code-Analyse

Die Dead Code-Analyse von AMELIO Logic Discovery geht jedoch noch einen Schritt weiter. Sowohl bei der Entwicklung von COBOL- als auch PL/I-Anwendungen werden Copybooks bzw. Includes oder, im Falle von ADS-Anwendungen, Makros verwendet. In diesen Fällen reicht die Analyse einzelner Programme nicht aus, denn der tote Code in den Programmen kann durch die Copybooks, Includes oder Markos entstanden sein. Darüber hinaus ist es auch möglich, dass diese Code enthalten, der nie kompiliert bzw. generiert wird. Solcher Code wird ebenfalls durch AMELIO Logic Discovery ermittelt. Um die Analyse auch für Copybooks und Includes zu ermöglichen, bildet AMELIO Logic Discovery die entsprechende Compile-Funktionalität nach. Das Ergebnis der Coverage Analyse enthält die folgenden Informationen:

Coverage Report of Macro HDLMSG030

Macro HDLMSG030 is
used by 10 primary source(s).
and called 40 times

		# gen.	# exec.
1	**PDL*201303131002/HDLMSG030/80/		
2	HDLMSG-#01.	40	32
3	.IF-01.OC./AC/DX/KL/OG/	40	
4	MOVE 'R-#01' TO MSGHDL-REQCODE.	40	32
5	CALL 'HDLMSG' USING MSGHDL-REC.	40	32
6	.IFELSE	0	
7	MOVE '#01' TO MSGHDL-REQCODE.	0	
8	CALL 'HDLMSG2' USING MSGHDL-REC.	0	
9	PERFORM CHK-HDL-MSG2-RET.	0	
10	.IFEND	40	
11	.IF-M1.NE.DONE	40	
12	INIT-REC.	10	10
13	INITIALIZE MSGHDL-REC.	10	10
14	.SL R-PROG	10	
15	CHK-HDL-MSG2-RET.	10	0
16	IF MSGHDL-RETCODE > 0	10	0
17	DISPLAY 'Message invalid: ', MSGHDL-MSGTEXT	10	0
18	GO TO STOP-RUN.	10	0
19	.SET-M1=DONE,	10	
20	.IFEND	40	

- Wie viele Primärsourcen das Copybook, den Include oder das Makro verwenden
- Wie oft das Copybook oder der Include bei der Kompilierungen bzw. das Makro bei Generierungen aufgerufen wurden
- Wie oft bestimmte Abschnitte eines Copybooks oder Includes bei einer Kompilierung bzw. eines Makros bei der Generierung verwendet wurden
- In wie vielen Fällen der erzeugte Code tatsächlich ausführbar ist

Codebereinigung

Nachdem der tote Code in all seinen Facetten ermittelt und dokumentiert wurde, kann er nun entfernt werden. Eine manuelle Bereinigung birgt allerdings die Gefahr, dass einige Zeilen nicht gelöscht oder zu viel gelöscht werden, so dass die Programmlogik verändert wird. Sicherer und schneller ist es daher die Transformationsfunktion von AMELIO zu nutzen. Diese entfernt den gesamten toten Code, Anweisungen ebenso wie Datendefinitionen.

Fazit

COBOL- und PL/I-Anwendungen sind über viele Jahre und Jahrzehnte gewachsen. Neben der wirklich notwendigen Funktionalität hat sich dabei auch eine Menge überflüssiger Ballast angesammelt, der die Wartung und das Verstehen der Anwendungen erschwert. AMELIO Logic Discovery hilft mit seinen Inventur-Funktionen einen Überblick darüber zu gewinnen, aus welchen Bestandteilen sich die Anwendung zusammen setzt und in welcher Beziehung diese zueinander stehen. Die Dead Code-Analyse ermittelt toten Code, sowohl tote Anweisungen als auch Datendefinitionen, und kann diese auch automatisch entfernen. Als Ergebnis erhält man aussagekräftige Dokumentationen sowie bereinigte Anwendungen, die wesentlich leichter zu warten und zu verstehen sind. AMELIO Logic Discovery-CleanUp unterstützt also bei der Wartung und der Qualitätssicherung.

AMELIO Logic Discovery - Frühjahrsputz für COBOL- und PL/I-Anwendungen

Der Frühling naht und die Sonne bringt Licht in Wohnungen und Häuser. Staub und Schmutz, die sich in der dunklen Jahreszeit ungesehen ansammeln konnten, werden sichtbar, so dass in vielen Haushalten nun ein Frühjahrsputz beginnt.
Warum den Frühjahrsputz nicht auch gleich auf Ihre COBOL- und PL/I-Anwendungen ausweiten?

2 ADS in Entwicklungsumgebungen von Micro Focus und IBM

Viele unserer Kunden, die mit ADS entwickeln, setzen gleichzeitig Entwicklungsplattformen von Micro Focus oder IBM ein. Diese bieten die Funktionalität zur integrierten Entwicklung von Großrechneranwendungen, wie z.B. die Entwicklung von COBOL und PL/I für IBM z/OS auf Arbeitsplatzrechnern unter Microsoft Windows. In enger Zusammenarbeit mit Micro Focus und IBM sind jetzt integrierte Lösungen für ADS und die Entwicklungsplattformen dieser Anbieter entstanden.

MICRO FOCUS®

Ready for IBM Rational. software

Durch die Konsolidierung der Entwicklungsumgebungen wird den Entwicklern eine einheitliche, produktive, grafische und moderne Arbeitsumgebung für die Entwicklung von Java- und (Web)Front-Ends sowie von COBOL-, PL/I- und Delta ADS-Anwendungen bereitgestellt.

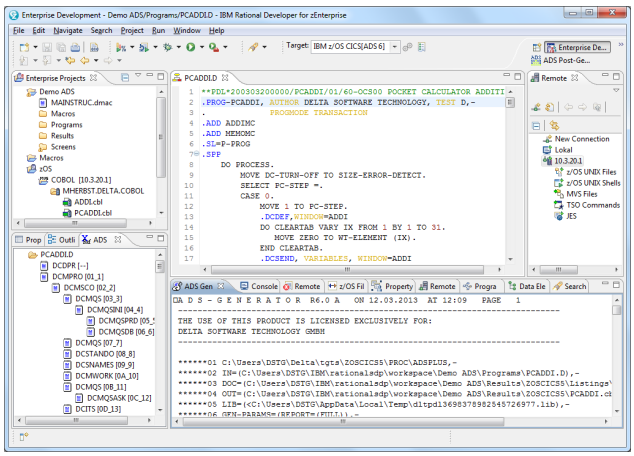
Dadurch bieten wir unseren Kunden jetzt, neben ADS on SCOUT² und ADS on Eclipse, auch ADS für weitere Entwicklungsumgebungen an:

- ADS on Micro Focus EDZ
  (Enterprise Developer for z)
- ADS on IBM RDz
  (Rational Developer for System z)

Beide Entwicklungsumgebungen basieren auf der Open Source-Plattform Eclipse. ADS on Eclipse integriert sich so beispielsweise in den Micro Focus

Workflow Manager, wodurch die Code-Generierung nun direkt als Teil des Workflows ausgeführt werden kann.

Durch die ADS-Anbindung an RDz können COBOL-Generate automatisch in ein Dataset auf dem z/OS-Host übertragen werden. Ebenfalls kann automatisch für diese Generate ein Compile/Link-Job auf dem Host gestartet werden.



Durch die Integration der Entwicklungswerkzeuge unter Eclipse ergeben sich folgende Vorteile:

- Einheitliche Oberfläche für ADS, COBOL, PL/I, C/C++, Java, ...
- Funktionalitäten moderner Entwicklungstechniken:
- Grafisches Debugging
- Real-Time Syntax-Check
- Syntax-Vervollständigung
- Smart Editor
- Weitere Integration mit Eclipse-basierenden Werkzeugen
- Anbindung von SCM Systemen wie z. B. Serena ChangeMan und ERO
- Lokale als auch remote Entwicklung
- u.v.m.

Wobei die Verlagerung der Entwicklung auf den PC folgenden Nutzen ergibt:

- Erhöhte Produktivität bei Erstellung und Wartung von Anwendungen
- Kürzere Entwicklungszyklen
- Entlastung des Mainframes durch MIPS-Einsparung
- Der Mainframe wird für Nachwuchskräfte attraktiver
- u.v.m.

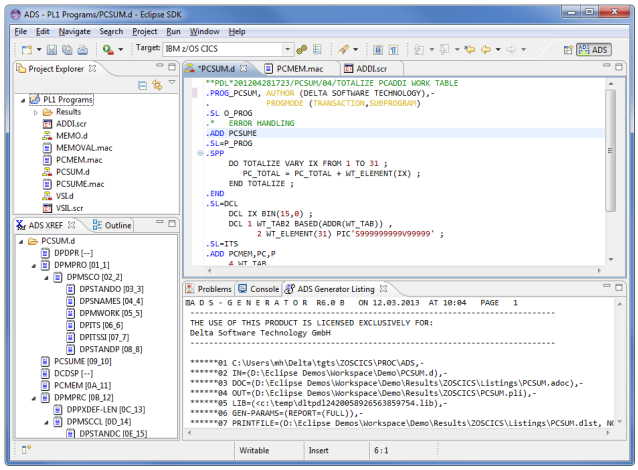
Durch die Integration von Werkzeugen und das Vermeiden von Medienbrüchen werden die Entwicklungs- und Testprozesse deutlich verkürzt.

3 ADS 6 for PL/I ist jetzt als Final Release verfügbar

ADS 6 for PL/I hilft mit seinen neuen Funktionen, die mit ADS generierten PL/I-Anwendungen und die mit ADS MACRO geschaffenen Anwendungs-Frameworks noch effizienter zu entwickeln, zu pflegen und zu testen:

- ADS 6 for PL/I verbessert das Verständnis der mit ADS entwickelten PL/I-Anwendungen.
- Spracherweiterungen für ADS MACRO vereinfachen die Entwicklung und Wartung der Anwendungs-Frameworks.

- Der Macro Optimizer sorgt für bessere Lesbarkeit und Wartbarkeit der bereits vorhandenen Macros.
- Der Post-Generation Debugger vereinfacht das Verständnis und den Test des Generierungsprozesses.
- Reports unterstützen die Wartung und helfen, die Anwendungen zu (re-)dokumentieren.
- Analysen liefern zuverlässige Informationen für die Bewertung der Anwendungen.



Mit dem Final Release ist ADS 6 for PL/I nun für alle Zielplattformen (Operating Systems, TP-Monitore und Datenbanken) verfügbar. Hier finden Sie eine Übersicht aller Zielplattformen.

4 Neues SCORE Release

Am 18. März 2013 wurde die neue Version 4.8 von SCORE Adaptive Bridges und SCORE Data Architecture Integration freigegeben. Kunden mit bestehenden Wartungsverträgen können das Update kostenlos anfordern.

Die Version 4.8 enthält die folgenden neuen Funktionen und Verbesserungen:
SCORE Adaptive Bridges

- Einfach zu erstellende Transaction Interfaces
- Definition von Tagged Values in den Generator-Einstellungen
- Performance-Steigerung bei der Generierung

SCORE Data Architecture Integration

- Einfach zu erstellende Transaction Interfaces
- Definition von Tagged Values in den Generator-Einstellungen
- Performance-Steigerung bei der Generierung
- Definition beliebiger Expressions in GROUP BY-Klauseln
- Unterstützung von Tabellen ohne Primary Keys und Verwendung von SQL Pseudo-Columns
- Komfortables (De)Aktivieren von Indikatorfeldern
- Übernahme von Kommentaren aus dem Data Definition File in das Composition Model

Weitere Informationen über die neuen Funktionen finden Sie in den Release Notes im Support-Bereich.

5 Vom Schuster mit den geraden Absätzen

[Daniela Schilling] Generatoren und domänenspezifische Sprachen (DSLs) stellen zusammen mit Modellen zwar die zentralen Elemente der modellgetriebenen Entwicklung dar, sie selber werden aber

mit den herkömmlichen Methoden paradoxerweise nicht modellgetrieben entwickelt. Daraus resultiert, dass die Generator- und DSL-Entwicklung als zeitaufwändige und fehleranfällige Geheimwissenschaft gilt. In diesem Artikel stelle ich eine Lösung vor, mit der dieses Paradoxon behoben werden kann und Aufwand und Komplexität deutlich reduziert werden können. Mehr dazu lesen Sie in der OBJEKTspektrum 06/2012.

Weitere Informationen

HyperSenses - Integriertes System für modellgetriebene Entwicklung von DSLs und Software-Generatoren.

Mehr Newsletter und unsere Newsletter-Verwaltung finden Sie unter: www.delta-software.com/newsletter

Bestellnummer: NL 11'013.01 – März 2013