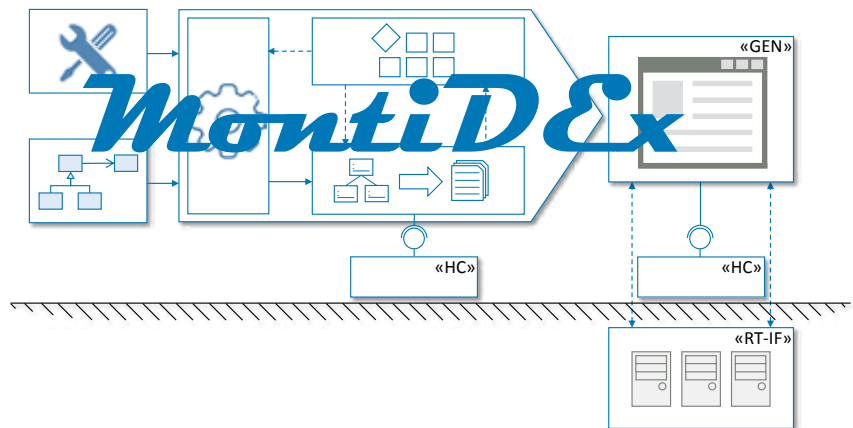


Alexander Roth

Adaptable Code Generation of Consistent and Customizable Data-Centric Applications with MontiDEx



Adaptable Code Generation of Consistent and Customizable Data-Centric Applications with MontiDEx

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

M.Sc. RWTH
Alexander Roth
aus Ujar, Russland

Berichter: Universitätsprofessor Dr. rer. nat. Bernhard Rumpe
Universitätsprofessor Dr. rer. nat. Albert Zündorf

Tag der mündlichen Prüfung: 15. November 2017

Aachener Informatik-Berichte, Software Engineering

herausgegeben von
Prof. Dr. rer. nat. Bernhard Rumpe
Software Engineering
RWTH Aachen University

Band 31

Alexander Roth
RWTH Aachen University

**Adaptable Code Generation of
Consistent and Customizable Data-Centric
Applications with MontiDEx**

Shaker Verlag
Aachen 2017

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Zugl.: D 82 (Diss. RWTH Aachen University, 2017)

Copyright Shaker Verlag 2017

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8440-5688-4

ISSN 1869-9170

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

Information systems are software systems that address current demands for harvesting, storing, and manipulating structured information [Kaj12]. A part of information systems are client applications (subsequently called *data-centric applications*) with a graphical user interface (GUI) to execute predefined CRUD (Create, Read, Update, and Delete) operations and display the managed data. The development and prototyping of such software systems contain common repetitive and error-prone development tasks such as implementing a *data structure*, which is a source-code-representation of the managed information, realizing a GUI, and providing access to a persistence infrastructure.

Model-driven development (MDD) aims to reduce the development effort of data-centric applications, improve software quality, and reduce development costs [BCW12] by exploiting platform-independent models as primary development artifacts [SVC06]. Each model is an instance of a domain-specific language (DSL), which enables a high-level and abstract description of (parts of) a software system using domain terminology. MDD tools systematically transform a model into executable source code.

Nevertheless, effective MDD of data-centric prototypes has to address several challenges. In particular, fully executable prototypes have to be generated including the data structure and GUI, which is necessary to support information gathering from end users [MFM+13]. Moreover, adequate DSLs have to be provided to describe structured information and facilitate underspecification that enables prototyping in early stages of the development. These concerns are additionally influenced by orthogonal customization concerns of the generated source code to support agile software development (cf. [BPRFF15]).

Similar holds for MDD of data-centric development, where the same challenges have to be addressed. In addition, an MDD approach has to avoid additional overhead introduced by maintaining DSLs and MDD tools (cf. [MK09]) to improve developer acceptance (cf. [KBR11]). Furthermore, such an approach has to provide adaptation mechanisms for MDD tools to facilitate MDD tool reuse by enabling framework-like and standalone use.

To address the aforementioned challenges, this thesis contributes methods and concepts to a lightweight and agile MDD approach for data-centric application development and its data structure prototyping. The goal is to improve development efficiency by reducing the necessary manual implementation effort for repetitive implementation tasks. In particular, implementing a GUI, a data structure that ensures data consistency, a connection to a persistence infrastructure, and support for process automation, which is considered as the automated execution of CRUD operations. This is achieved by a language family for structural and behavioral modeling, code generators, and lightweight methods. Since such an approach is always accompanied by customization, adaptation, and extension concerns (cf. [SVC06]), mechanisms for manually-written customizations

of generated source code, as well as a modular and adaptable code generator design to facilitate code generator reuse are addressed (cf. [ZR11]). Furthermore, a method and technical realization for all proposed concepts is provided by the MontiCore Data Explorer (MontiDEx) code generator.

Employing the methods proposed in this thesis improves development and prototyping of data-centric applications by providing a unified set of languages and lightweight methods abstracting from implementation details and supporting customization and adaptation concerns of generated source code and MDD tools.

Kurzfassung

Informationssysteme sind Softwaresysteme, die das Verwalten von strukturierten Daten unterstützen und fördern (vgl. [Kaj12]). Ein Teil solcher Informationssysteme sind Client-Anwendungen (im weiteren Verlauf *datenzentrische Anwendungen* genannt). Diese besitzen eine graphische Benutzeroberfläche und erlauben das Ausführen von CRUD (Create, Read, Update und Delete) Operationen auf den verwalteten Daten, sowie das Anzeigen der Resultate. Jedoch beinhaltet die Entwicklung und das Prototyping solcher datenzentrischen Anwendungen viele gemeinsame und sich wiederholende Entwicklungsaufgaben, wie z.B. das Implementieren einer Datenstruktur, die die Quellcode-Repräsentation der verwalteten Information darstellt, das Realisieren einer graphischen Benutzeroberfläche und das Anbieten einer Schnittstelle zu einer Persistenzinfrastruktur für die Datenspeicherung.

Ein Ansatz zur Reduzierung dieser sich wiederholenden Entwicklungsaufgaben zur Steigerung der Softwarequalität und zur Senkung der Entwicklungskosten bietet die Modellgetriebene Softwareentwicklung (MDD) (vgl. [BCW12]). Dies geschieht dadurch, dass plattformunabhängige Modelle als primäres Entwicklungsartefakt genutzt werden (vgl. [SVC06]). Jedes Modell ist dabei eine Instanz einer domänenspezifischen Sprache (DSL), welche ein hohes Abstraktionsniveau bietet um Teile oder ganze Softwaresysteme zu beschreiben und gleichzeitig Domänenterminologie zu verwenden. Werkzeuge transformieren systematisch ein Modell in ausführbaren Quellcode.

Dennoch muss ein effektiver Ansatz zur modellgetriebenen Softwareentwicklung von datenzentrischen Prototypen mehrere Herausforderungen bewältigen. Eine Herausforderung ist die Generierung von vollständig ausführbaren Prototypen. Dies ist Notwendig um die Informationsgewinnung von Endbenutzern zu unterstützen (vgl. [MFM+13]). Diese ausführbaren Prototypen müssen die Datenstruktur beinhalten und eine graphische Benutzeroberfläche anbieten. Bei der modellgetriebenen Softwareentwicklung setzt dies jedoch voraus, dass geeignete DSLs zur Verfügung gestellt werden, die es ermöglichen strukturierte Informationen abstrakt zu beschreiben und auch während der frühen Phase des Prototypings, welche auch Unterspezifikation beinhaltet, genutzt werden können. Eine weitere Herausforderung ist die Bereitstellung von Möglichkeiten zur Anpassung der generierten datenzentrischen Anwendung, die vor allem in einer agilen Entwicklungsumgebung erforderlich sind (vgl. [BPRFF15]).

Gleiche Herausforderungen existieren auch in der modellgetriebene Softwareentwicklung von datenzentrischen Anwendungen. Jedoch sollte zusätzlich der gewählte MDD Ansatz Overhead vermeiden, der durch die Wartung und die Entwicklung von DSLs und geeigneten Werkzeugen eingeführt wird (vgl. [MK09]). Die Vermeidung dieses Overheads steigert die Akzeptanz der Entwickler (vgl. [KBR11]). Darüber hinaus muss ein derartiger Ansatz Anpassungsmechanismen für die genutzten Werkzeuge bereitstellen, um deren Wiederverwendung zu fördern.

Um diese Herausforderungen zu adressieren, werden in dieser Arbeit leichtgewichtige Methoden und Konzepte für ein modellgetriebenes Datenstruktur-Prototyping und die modellgetriebene Softwareentwicklung von datenzentrischen Anwendungen vorgestellt. Ziel ist es, die Effizienz in der Entwicklung zu verbessern, indem der notwendige manuelle Implementierungsaufwand für sich wiederholende Implementierungsaufgaben reduziert wird. Dabei steht insbesondere der Implementierungsaufwand bei der Entwicklung einer graphischen Benutzeroberfläche, einer Datenstruktur und einer Anbindung an eine Persistenzinfrastruktur im Fokus dieser Arbeit. Dies wird durch eine geeignete Sprachfamilie für die Struktur und Prozessbeschreibung von datenzentrischen Anwendungen, geeignete Code-Generatoren und leichtgewichtige Entwicklungsmethoden erreicht. Um Anpassungen im generierten Produkt als auch im Generator zu ermöglichen, werden Methoden zur handgeschriebenen Erweiterung und Konzepte zur Realisierung eines modularen Code-Generators vorgestellt, welche die Wiederverwendung fördern (vgl. [ZR11]). Die entwickelten Konzepte wurden im MontiCore Data Explorer (MontiDEx) Code Generator, sowie dem MontiDEx Produkt realisiert.

Danksagung

In der Entstehungszeit meiner Promotion wurde ich von vielen lieben Menschen unterstützt und begleitet, die durch hilfreiche Diskussionen, tatkräftige Unterstützung und notwendige Ablenkung zum Erfolg dieser Promotion beigetragen haben. Deshalb möchte ich an dieser Stelle diesen Menschen meinen tiefsten Dank aussprechen.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr. Bernhard Rumpe für die Betreuung dieser Promotion und die spannende und abwechslungsreiche Zeit am Lehrstuhl. Die vielen konstruktiven und lebhaften Diskussionen sowie die vielen Ratschläge haben den Erfolg dieser Promotion entscheidend bewirkt. Auch danke ich dir sehr, Bernhard, dass du mir die Möglichkeit gegeben hast frühzeitig die Leitung verschiedener Forschungs- und Industrieprojekte zu übernehmen und mich so in meiner persönlichen Entwicklung unterstützt hast. Weiterhin bedanke ich mich bei dir für dein Vertrauen mir die Gruppenleitung der Digitalisierungsgruppe zu übergeben.

Weiterer Dank gilt Prof. Dr. Albert Zündorf für das Interesse an meiner Arbeit, das sehr gute Feedback und die Übernahme meines Zweitgutachtens. Ebenfalls möchte ich mich bei Prof. Dr. Ulrik Schröder für die Leitung meines Promotionskomitees und die Abnahme der Prüfung in der praktischen Informatik bedanken. Herrn apl. Prof. Dr. Thomas Noll danke ich für die Abnahme der Prüfung in der theoretischen Informatik.

Herzlich bedanke ich mich auch bei allen Kollegen, die mich während meiner Zeit am Lehrstuhl (oftmals auch außerhalb) begleitet haben. Insbesondere möchte ich Dr. Martin Schindler für sein Mentoring am Anfang meiner Promotionszeit danken. Ebenfalls danke ich Antonio Navarro Pérez für die viele Ideen und den großartigen Einfluss auf meine Promotion. Mein besonderer Dank gilt meinem ehemaligen Gruppenleiter Dr. Pedram Mir Seyyed Nazari. Wir haben zusammen den Weihnachtsmann überlebt und konnten immer über jegliche Themen lebhaft und lange diskutieren. Danke für die gemeinsame Zeit, die Zusammenarbeit in der Forschung und Lehre und die Motivation während meiner Promotion! Dr. Klaus Müller danke ich für die sehr gute Zusammenarbeit an verschiedenen Publikationen und die vielen gemeinsamen Forschungsworkshops. Auch wenn du als externer selten anwesend warst, war die Zusammenarbeit mit dir sehr fruchtbar und erfolgreich. Herrn “von Wenckstern” danke ich für seine offene, lustige und immer unterhaltsame Art mit der er das Leben am Lehrstuhl aufgeheitert hat. Gleichzeitig danke ihm auch für die sehr intensiven und hilfreichen Diskussionen. Evgeny Kusmenko möchte ich für seine Motivation zur sportlichen Betätigung danken, aber auch für die gute Zusammenarbeit. Ich bedanke mich auch sehr bei Kai Adam, der mich stark in der letzten Phase meiner Promotion und in unterschiedlichen Forschungs- und Industrieprojekten tatkräftig unterstützt hat.

Darüber hinaus danke ich Sylvia Gunder und Gabriele Heuschen, die mich bei allen organisatorischen Aufgaben unterstützt haben und mein Anlaufpunkt bei Unklarheiten waren. Ebenfalls bedanke ich mich bei Galina Volkova und Marita Breuer, die im Rah-

men des MontiCore Projektes ihr Wissen gerne geteilt haben und mich auch während vieler Demonstrationen und Implementierungsarbeiten unterstützt haben. Außerdem danke ich allen Wissenschaftlichen Mitarbeitern des Lehrstuhls: Vincent Bertram, Arvid Butting, Robert Eikermann, Timo Greifenberg, Dr. Arne Haber, Robert Heim, Steffen Hillemacher, Lars Hermerschmidt, Andreas Horst, Katrin Hölldobler, Oliver Kautz, Carsten Kolassa, Thomas Kurpick, Achim Lindt, Matthias Markthaler, Dimitri Plotnikov, Deni Raco, Dr. Jan Oliver Ringert, Steffi Schrader, Christoph Schulze und Dr. Andreas Wortmann. Für die tatkräftige Unterstützung bedanke ich mich auch bei allen Auszubildenden: Lennart Bucher, Manuel Pütz, Jerome Pfeifer, Ben Mainz, Brian Sinkovec und Max Voss. Vielen Dank auch an all diejenigen, die meine Arbeit Korrektur gelesen haben: Kai Adam, Arvid Butting, Robert Heim, Oliver Kautz, Matthias Markthaler, Dr. Klaus Müller, Dr. Pedram Mir Seyed Nazari, Max Voß und Michael von Wenckstern.

Natürlich gilt mein Dank auch allen Studenten und Studentinnen, die an der Promotion in Form von studentischen Abschlussarbeiten tatkräftig mitgewirkt haben: Aydin Alatas, Denis Domm, Ralph Geerkens, Thomas Maiwald, Patrick Jan Schlesiona, Dominik Studer, Philip Uhl und Enis Zejnilovic. Insbesondere danke ich Junior Lekane Nimpa für die lange Zusammenarbeit an MontiDEx und die sehr guten Implementierungs- und Gestaltungsarbeiten.

Abschließend gilt mein unendlicher Dank meinen Eltern und meinen Freunden, die mich auf meinem bisherigen Lebensweg begleitet und unterstützt haben. Liebe Eltern, Peter und Katharina Roth, ich danke euch von ganzem Herzen. Ihr habt mir die Motivation und das nötige Durchhaltevermögen gegeben und als Vorbilder gezeigt, wie man seine Ziele erreichen kann. Diese Promotion wäre ohne eure Aufopferung, Liebe und Unterstützung nicht möglich gewesen. Ebenfalls bedanke ich mich auch meiner Schwester, Olga Knabe, die mir auch während schwieriger Zeiten immer mit Rat und Tat zur Seite stand, mich immer unterstützt und mit kleinen Geschenken motiviert hat. Auch möchte ich Dr. Bastian Knabe und Melissa Anastasia Knabe danken für die Aufheiterung und das Interesse an meiner Promotion. Ich danke auch Michelle Egge für die grenzenlose Unterstützung und unglaubliche Geduld. Für die notwendige Ablenkung und Erholung möchte ich euch, Hauke und Lisa Schaper, ganz herzlich danken. Schließlich danke ich auch Florian Kerber, Michael Lutz und Dr. Andreas Ganser für die Begleitung auf meinem bisherigen Lebensweg und die schöne Zeit mit euch.

Aachen, November 2017
Alexander Roth

Contents

1	Introduction	1
1.1	Context of the Thesis	4
1.2	Objectives and Contribution	4
1.3	Organization of the Thesis	6
1.4	Related Own Publications	8
2	Foundations: Model-Driven Development and Data-Centric Application	9
2.1	Model-Driven Development	9
2.1.1	Domain-Specific (Modeling) Language	10
2.2	MontiCore Language Workbench and Code Generation Framework	11
2.2.1	MontiCore Grammar	11
2.2.2	AST Generation from MontiCore Grammars	13
2.2.3	Symbol Table	14
2.2.4	Code Generation	15
2.3	Data-Centric Applications	18
2.3.1	Layered Architecture	19
2.4	Related MDP and MDD Approaches	19
2.4.1	Related MDP Approaches for Data-Centric Applications	19
2.4.2	Related MDD Approaches for Data-Centric Applications	21
3	Requirements for the Envisioned Methodology	27
3.1	Typical Scenario for Generative Development of a Data-Centric Application	27
3.1.1	Model-Driven Prototyping of Data Structures	28
3.1.2	Model-Driven Development of Data-Centric Applications	30
3.1.3	Roles in the Development and Prototyping Process	32
3.2	Primary High-Level Requirements	33
3.2.1	General Requirements	34
3.2.2	Modeling Requirements	36
3.2.3	Code Generator Requirements	37
3.2.4	Generated Product Requirements	38
3.3	Envisioned Methods for MDP and MDD of Data-Centric Applications . .	38
3.3.1	MDP of Data Structures with MontiDEx	40
3.3.2	MDD of Data-Centric Applications with MontiDEx	41

4	UML Class Diagrams in Analysis, Design and Implementation	45
4.1	Analysis, Design, and Implementation Model	45
4.1.1	Language Concepts in Analysis Models	47
4.2	CD4A: Modeling Language for Analysis Models	47
4.2.1	Model Definition	48
4.2.2	Interfaces, Classes, and Enumerations	49
4.2.3	Attributes and Predefined Data Types	50
4.2.4	Associations	51
4.2.5	Context Conditions	55
4.3	CD4Code: Modeling Language for Implementation Models	62
4.3.1	Modifiers	63
4.3.2	Constructor-Signatures	63
4.3.3	Method-Signatures	64
4.3.4	CD4Code Interface	64
4.3.5	CD4Code Enumeration	65
5	Systematic CD4A ML to a Java Mapping	67
5.1	General Considerations and Mapping Guidelines	67
5.2	Mapping of CD4A Concepts to Java Source Code	69
5.2.1	Mapping CD4A Model Definition	69
5.2.2	Mapping CD4A Interfaces	70
5.2.3	Mapping CD4A Classes	71
5.2.4	Mapping CD4A Enumerations	73
5.2.5	Mapping CD4A Attributes	74
5.2.6	Mapping CD4A Unidirectional Associations	77
5.2.7	Mapping CD4A Bidirectional Associations	81
5.2.8	Mapping CD4A Ordered Associations	84
5.2.9	Mapping CD4A Qualified Associations	85
5.2.10	Mapping CD4A Derived Associations	88
5.2.11	Mapping CD4A Compositions	90
5.3	Method for Handling Mandatory-to-Mandatory Associations	91
6	Generated Code Customization via Handcoded Extensions and Hot Spots	95
6.1	General Considerations of Handcoded Extensions	96
6.1.1	Separation of Generated and Non-Generated Artifacts	96
6.1.2	Override-Static-Pattern	97
6.2	Integration of Generated and Non-Generated Code	99
6.2.1	Implementation of Interface Extensions using Java-Default Interfaces	101
6.2.2	CD4A Hierarchy and Handcoded Extensions	102
6.3	Customization via Hot Spots in Generated Source Code	103

6.4	Methods for using Handcoded Extensions	104
6.4.1	Extending and Associating External Data Types in CD4A Models	106
7	A Customizable Data-Centric Infrastructure	109
7.1	General Considerations and Architectural Design Drivers	110
7.1.1	Architectural Impact of Infrastructure Customization	112
7.1.2	Type-specific Method Invocation via Double Dispatching	112
7.1.3	Run-time Environment and Modularity	113
7.2	Mapping CD4A Models to an Application Layer	114
7.2.1	Object Instantiation and Manipulation	114
7.2.2	Data Structure Management	117
7.3	Mapping CD4A Models to a Presentation Layer	120
7.3.1	Mapping Model Definition, Interfaces, Classes, and Enumerations	121
7.3.2	Technical Realization of GUI Architecture	126
7.3.3	Manipulating Objects via Model-Specific Commands	129
7.3.4	Managing Execution of Model-Specific Commands	130
7.4	Generic Persistence Infrastructure	133
7.4.1	Generic CD4A Meta-Model	133
7.4.2	Multi-Tenancy and Role-Base Access Control	135
7.4.3	Technical Realization of Accessing the WebService	137
7.5	Mapping CD4A Models to a Persistence Layer	140
7.5.1	Lazy Loading of Objects from the Persistence Infrastructure	140
7.6	Method for Consistent Data Migration	141
8	Synergetic Transformation- and Template-based Code Generation	145
8.1	General Requirements	146
8.2	Integration of Transformation- and Template-based Code Generation	147
8.2.1	Case Example: Statecharts-to-Java Source Code	149
8.2.2	An Object-Oriented Intermediate Representation using CD4Code	150
8.2.3	Model-to-Model Transformations	151
8.2.4	Adding Implementation Details via Template Attachments	154
8.2.5	Model-To-Text Transformation	155
8.3	Template Adaptation via Template Hook Points and Template Extensions	157
8.3.1	Adaptation via Template Hook Points	157
8.3.2	Adaptation via Template Extensions	158
8.3.3	Technical Realization in MontiCore	160
8.4	Methods for Transformation Design and Management	162
8.4.1	Method for Transformation and Template Development	162

9	MontiDEx: MontiCore Data Explorer Code Generator	165
9.1	Technique to Handle Underspecification in MontiDEx	166
9.1.1	CD4A Underspecification and Defaults	167
9.2	MontiDEx Architecture and Technical Realization	168
9.2.1	Technical Realization of the Common Infrastructure	168
9.3	Methods for Code Generator Configuration	173
9.3.1	Technical Realization of MontiDEx Configurations	174
9.4	MontiDEx Reporting Facility	177
9.4.1	Textual Reports	177
9.4.2	Graphical Report	179
9.5	Method for Adapting and Deploying MontiDEx	180
9.5.1	Method for Adapting the MontiDEx Code Generator	181
9.5.2	MontiDEx Project Types and Deployment	181
10	Case Example: Extended Infrastructure for Process Automation	183
10.1	General Considerations and Requirements	184
10.2	ADJava: Activity Diagram Modeling Language	185
10.2.1	Activity Definition	186
10.2.2	Actions	188
10.2.3	Object Nodes	189
10.2.4	Control And Object Flow	190
10.2.5	Roles	193
10.2.6	Pin and Type Auto-Connect	193
10.3	Execution of ADJava Models	196
10.3.1	Method for Interpretation of ADJava Models	198
10.3.2	Code Generation from ADJava Models	200
10.3.3	Technical Realization of the Extended Data-Centric Infrastructure	204
10.3.4	Technical Realization of the MontiDEx Code Generator Extension	205
10.4	Method for Developing Processes with ADJava	207
10.5	Evaluation and Limitation	208
10.5.1	Evaluation of MontiDEx Customization and Adaptation Approaches	208
10.5.2	Limitations	209
11	Case Example: MDP and MDD with MontiDEx	211
11.1	Points-of-Interest Management System	211
11.1.1	Technical Realization	212
11.1.2	Discussion	215
11.2	Audio and Video Streaming Platform	215
11.2.1	Technical Realization	217
11.2.2	Discussion	220

11.3 Examination Regulation System	223
11.3.1 Technical Realization	224
11.3.2 Discussion	225
12 Conclusion	227
12.1 Summary	227
12.2 Potential Future Work	229
Bibliography	231
A Index of Abbreviations	255
B Diagram and Listing Tags	257
C Grammars	259
C.1 CD4Code Grammar	259
C.2 Activity Diagram Language Grammar	261
C.3 Activity Diagram Language Grammar with Embedded Java	264
D Examples	265
D.1 CD4A Model for Banking System	265
D.2 CD4A Model for the POI Management System	267
D.3 CD4A Model for the Audio and Video Streaming	268
D.4 CD4A Model for the Examination Regulation System	270
D.5 Activity Diagram for Transaction Submission	275
E Context Conditions	277
E.1 CD4A Context Conditions	277
E.2 CD4Code Context Conditions	283
E.3 Activity Context Conditions	284
F MontiDEx Hot Spots	295
F.1 Graphical User Interface	295
F.2 Application Core	307
F.3 Persistence	307
G MontiDEx Package Structure	309
H MontiDEx Hook Points	311
I Curriculum Vitae	313
List of Figures	315

Listings	321
List of Tables	325