

Stefan Hennig

Design of Sustainable Solutions for Process Visualization in
Industrial Automation with Model-Driven Software Development

Beiträge aus der Automatisierungstechnik

Stefan Hennig

**Design of Sustainable Solutions for Process
Visualization in Industrial Automation with Model-
Driven Software Development**

 VOGT

Dresden 2012

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen
Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
<http://dnb.ddb.de> abrufbar.

Bibliographic Information published by the Deutsche Bibliothek
The Deutsche Bibliothek lists this publication in the Deutsche
Nationalbibliografie; detailed bibliographic data is available in the internet at
<http://dnb.ddb.de>.

Zugl.: Dresden, Techn. Univ., Diss., 2012

Die vorliegende Arbeit stimmt mit dem Original der Dissertation
„Design of Sustainable Solutions for Process Visualization in Industrial
Automation with Model-Driven Software Development“ von Stefan Hennig
überein.

© Jörg Vogt Verlag 2012
Alle Rechte vorbehalten. All rights reserved.

Gesetzt vom Autor

ISBN 978-3-938860-49-6

Jörg Vogt Verlag
Niederwaldstr. 36
01277 Dresden
Germany

Phone: +49-(0)351-31403921
Telefax: +49-(0)351-31403918
e-mail: info@vogtverlag.de
Internet : www.vogtverlag.de



**TECHNISCHE
UNIVERSITÄT
DRESDEN**

Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

**Entwurf nachhaltiger Lösungen zur
Prozessvisualisierung in der industriellen
Automatisierungstechnik mittels
modellgetriebener Softwareentwicklung**

Design of Sustainable Solutions for Process Visualization
in Industrial Automation with
Model-Driven Software Development

Stefan Hennig

Der Fakultät Elektrotechnik und Informationstechnik
der Technischen Universität Dresden

zur Erlangung des akademischen Grades eines

Doktoringenieurs

(Dr.-Ing.)

genehmigte Dissertation



Vorsitzender: Prof. Dr.-Ing. Ralf Lehnert

Gutachter: Prof. Dr. techn. Klaus Janschek
Prof. Dr. Kris Luyten

Tag der Einreichung: 02.03.2012

Tag der Verteidigung: 07.06.2012

Acknowledgments

Successful research has at least three preconditions: (1) a reliable and carefree financial situation, (2) passionate scientific advice, and (3) many enthusiastic supporters. To meet the first requirement, this work was funded as “Landesinnovationspromotion” by the *European Social Fund* and the *Free State of Saxony*. This funding gave me the independence to research freely from conflicts of interests.

Prof. Dr. techn. Klaus Janschek, Managing Director of the Institute of Automation, Technische Universität Dresden, ensured the second requirement for successful research. I would like to thank him for his guidance and for the resources which made this work possible. With a balanced relationship between *encouragement* and *challenge*, he left me all the creative freedom while he ensured a successful conclusion.

Prof. Dr. Kris Luyten, Professor at the Expertise Centre for Digital Media, Universiteit Hasselt, Belgium, invited me to a research stay in his team. During this short time, we implemented several exciting ideas and I was able to learn a lot about the real business of research. Particularly, Kris’ passion for science and for motivating support inspired me.

This work was realized as part of the research of the *Teleautomation* working group. It is based on collaborative work and a great number of shared ideas. Many thanks go to the fabulous team forming that group, particularly to the team leader *PD Dr.-Ing. Annerose Braune*. She asked countless critical questions and always had an open mind about unconventional approaches. I would like to thank *Evelina Koycheva*, *Matthias Freund*, and *Henning Hager* for the strong company and all the fruitful discussions. Henning had accompanied my project over several years at the narrowest. No one else has had such deep insight into my work.

The third requirement was met, on one side, by many students. Some parts of this work were greatly influenced by them, other parts were collaboratively developed. In particular, many thanks are devoted to *Nicolas Dingeldey*, *Alexander Witkowski*, *Ying Su*, and *Martin Halfter*. On the other hand, I am very grateful to the secretary of the institute, *Petra Möge*, and to the institute’s backbone, *Matthias Werner*, for all the things they arranged for me during the

past six years. Furthermore, I want to thank all the other doctorate students of the institute for having such a great time. Particular thanks go to *Thomas Kaden* who helped me a lot with preparing the doctoral viva.

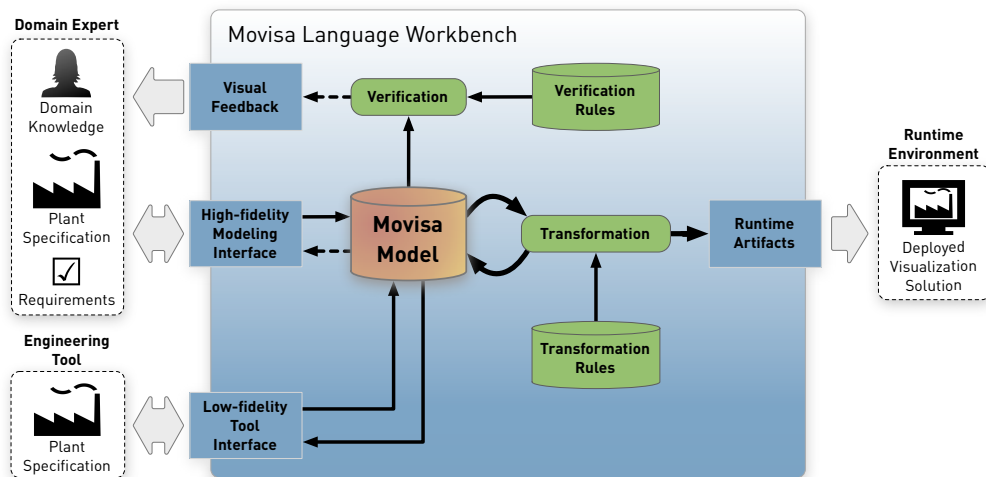
More important than getting technical support, however, is a strong support in “real life”. Hence, I would like to thank my friends for always supporting me during all the years of hard work. Special thanks go to *Maik Biebl* for being so patient with me and for his insatiable interest in my work despite not understanding a single word. Special thanks also go to *Bärbel Wöhlke* for polishing my English.

“What you have started, must also be brought to an end!” As I grew up under this premise, my *parents*, Lorita and Günter, and my *brother* Sebastian have always given me the courage to finish what I had started and the freedom to make my own decisions. I am in great debt to them.

And finally, I thank *Katrin Holinski* for reminding me so often that there is a life beyond technology.

Abstract

Industrial facilities are supervised using dedicated *Supervisory Control and Data Acquisition* (SCADA) applications. These applications, however, suffer from being developed using platform specific terminologies which cause that their operative characteristics are strongly merged with aspects of the technical realization. Platforms executing these applications are characterized by short innovation cycles, thus, decreasing the life time of SCADA applications. Industrial facilities, however, are required to be in operation for decades which possibly requires repeated redevelopment of these applications even if the operative characteristics remain the same. *Model driven* techniques are promising design approaches to foster sustainability of SCADA applications: They separate operative characteristics from their technical realization using *Domain Specific Languages*.



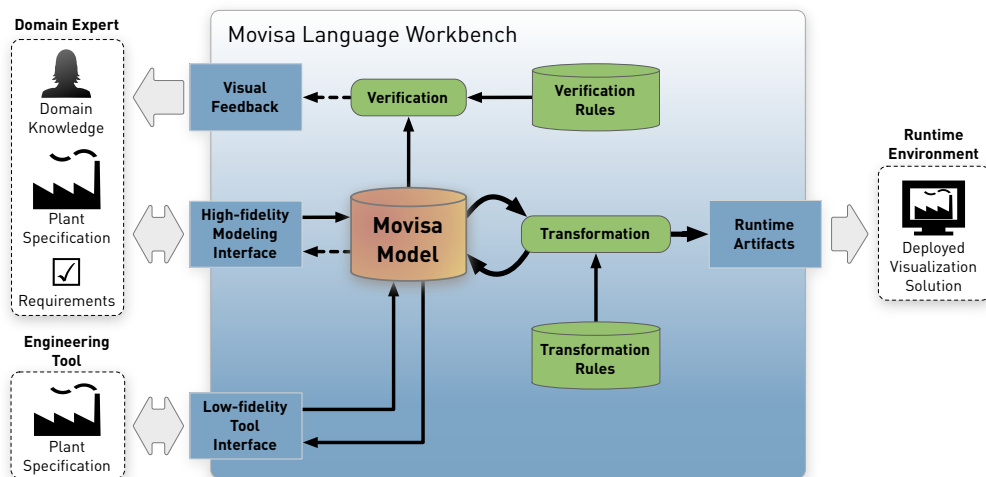
This thesis proposes the domain specific modeling workbench MOVISA. Its core consists of a domain specific modeling language enabling to capture operative characteristics of SCADA applications. For this purpose, it contains building blocks to create user interfaces, process data and communication relationships with automation specific data servers, and it allows to express custom functionality through an Executable UML realization. *Language Constraints* and model-integrity checks allow to identify errors in early design stages and ensure the correctness of models. Transformation rules capture aspects of the technical realization: They allow to process MOVISA models either to modify

these models or to automatically create runtime artifacts. In this context, different kinds of transformations are provided in order to support modelers in their assignments and, thus, to reduce the overall development effort. This complexity is encapsulated behind a high-fidelity modeling interface to be exploited by domain experts. It allows to solve problems with a common terminology that is very close to the respective solution space. Furthermore, engineering tools are able to populate MOVISA models via the low-fidelity tool interface.

Case studies from different fields of the domain *production automation* prove the language to be able to describe SCADA applications, thus, meeting related requirements of industrial automation. Sustainability of these applications can be ensured, among others, through automatic transformations, by reusing models and transformations in future projects and through having only one tool to master. The quintessence of this thesis is that even though model driven approaches are challenging with respect to provide effective tool environments, they are very promising means for creating sustainable software designs.

Kurzfassung

Unter *Supervisory Control and Data Acquisition* (SCADA) wird das Überwachen und Bedienen technischer Produktionsprozesse verstanden. Handelsübliche SCADA-Systeme erzeugen Lösungen auf Basis plattformspezifischer Terminologien. Daraus folgt eine enge Verzahnung funktionaler Inhalte mit Aspekten ihrer technischen Realisierung. Plattformen, auf denen SCADA-Lösungen genutzt werden, entstammen zunehmend dem Endverbrauchermarkt und unterliegen damit einer hohen Innovationsrate. Sich ändernde Plattformeigenschaften ziehen selbst bei gleichbleibenden funktionalen Inhalten eine Neuentwicklung dieser Lösungen nach sich. Die Einsatzzeit der SCADA-Lösungen ist somit verglichen mit der der industriellen Anlagen gering. Die modellgetriebene Software-Entwicklung bietet einen vielversprechenden Ansatz zur Erzeugung nachhaltiger SCADA-Lösungen, indem sie funktionale Inhalte von Aspekten ihrer technischen Realisierung auf Basis *Domänenspezifischer Sprachen* erlaubt.



Diese Arbeit schlägt zur Lösung der genannten Problemstellung die domänen-spezifische Werkzeugkette MOVISA vor. Zentraler Bestandteil ist eine domänen-spezifische Modellierungssprache, die funktionale Inhalte von SCADA-Lösungen zu beschreiben vermag. Dazu stellt sie Sprachmittel für die Beschreibung der Benutzungsschnittstelle sowie der Prozessdaten und Kommunikationsbeziehungen zur Verfügung. Aufgrund der Vielfalt technischer Prozesse enthält sie außerdem eine *Executable UML*-Realisierung, um die damit verbundenen Anforderungen zu adressieren. Mittel der Modellverifikation und -integritätsprüfungen ermöglichen

die Identifizierung von Fehlern bereits in frühen Entwurfsphasen und garantieren die Korrektheit der Modelle. Transformationsregeln enthalten die Aspekte der technischen Realisierung. Hinsichtlich einer automatischen Erzeugung der Laufzeitartefakte werden diese den MOVISA-Modellen zugeführt. Weitere Transformationen verarbeiten MOVISA-Modelle, um den Modellierer in seinen Aufgaben zu unterstützen und damit den Entwicklungsaufwand zu reduzieren. Die mit diesen Komponenten verbundene Komplexität bleibt dem Modellierer durch einen *high-fidelity* Arbeitsraum im vorgeschlagenen Werkzeug verborgen. Dies ermöglicht das Arbeiten mit einer Terminologie, die sich nah am Lösungsraum des jeweiligen Problems befindet. Über eine *low-fidelity* Schnittstelle erhalten Engineering-Werkzeuge Zugriff auf das MOVISA-Modell.

Fallstudien aus verschiedenen Anwendungsfeldern der Domäne *Produktionsautomatisierung* belegen, dass die vorgeschlagene domänenspezifische Sprache imstande ist, SCADA-Lösungen zu beschreiben. Die Nachhaltigkeit dieser Lösungen ist unter anderem durch automatische Transformationen, durch Wiederverwendung der Modelle und Transformationen in späteren Projekten sowie durch die Pflege nur eines Werkzeugs sichergestellt. Als Quintessenz dieser Arbeit wird festgestellt, dass modellgetriebene Ansätze zur Softwareentwicklung zwar vor dem Hintergrund der Bereitstellung effizienter Werkzeuge herausfordernd sind. Doch zeigen sie sich vielversprechend für den Entwurf nachhaltiger Softwarelösungen.

Contents

Acknowledgments	vii
Abstract	ix
Kurzfassung	xi
Contents	xv
List of Figures	xxvii
List of Tables	xxix
List of Listings	xxxi
Nomenclature	xxxiv
1 Introduction	1
1.1 Problem Statement and Aims	3
1.2 Thesis Structure	6
2 State of the Art	9
2.1 Human Machine Interface Engineering in Automation	9
2.2 Model Based User Interface Development	12
2.3 Model Driven Software Development	15
2.4 Domain Specific Languages	18
2.5 Requirements and Contributions	23
3 Language Model	31
3.1 Analyzing the Target Domain	31
3.1.1 Scripting Language	32
3.1.2 Process Data	33
3.1.3 User Interface Components	35

xiii

3.2	Core Language Model	40
3.2.1	Algorithm Model	41
3.2.2	Client Data Model	45
3.2.3	Presentation Model	52
3.3	Language Model Constraints	64
3.3.1	Intra-Submodel Constraints	65
3.3.2	Inter-Submodel Constraints	69
3.3.3	Model-Integrity Checks	70
3.4	Language Behavior Definition	72
3.4.1	Process Communication	73
3.4.2	Alarm Management	75
3.4.3	Specific Requirements	77
3.4.4	Reflecting Process States and Intervention	77
3.5	Conclusions	80
4	Concrete Syntax Notation	83
4.1	Preliminary Consideration	83
4.2	Algorithm Model: Pure Graphical Syntax Notation	84
4.3	Client Data Model: Pure Textual Syntax Notation	88
4.4	Presentation Model: Combined Syntax Notation	90
4.5	Conclusions	91
5	Movisa: Domain Specific Modeling Workbench	95
5.1	Requirements	96
5.2	Model Verification	97
5.3	Model Transformation	98
5.3.1	Dimensions of Code Generation	98
5.3.2	Specific Characteristics of Code Generation	99
5.3.3	On the Deployment of Runtime Artifacts	104
5.3.4	Evaluation of Existing Code Transformation Engines	106
5.4	On the Verification of the Language Behavior	107
5.5	Concrete Syntax Implementation	108
5.6	Movisa Modeling Workbench	108
5.7	Conclusions	110
6	Evaluation	111
6.1	Functional Requirements Evaluation	111
6.1.1	Case Study: Process Industries	112
6.1.2	Case Study: Autonomous Robots in Factory Automation	121

6.1.3	Case Study: Energy Supply Systems	125
6.1.4	Case Study: Health Care	128
6.2	Effectiveness Factors Evaluation	131
6.2.1	Reduce Solution Viscosity	131
6.2.2	Addressing a New Problem	133
6.2.3	Power in Combination	133
6.3	Conclusions	134
7	Exploiting Movisa in Supplementary Model-Based Environments	137
7.1	autoHMI: HMI Generation in Process Industries	137
7.2	Useware Engineering Process	139
7.3	Flepr: Flexible Transformation Based Workflows	141
7.4	Conclusions	145
8	Conclusions	147
8.1	Achievements and Contributions	147
8.2	Future Work	153
	References	157
	Appendix A Sample Visualization Applications	I
A.1	Power Supply Network Monitoring in the Technische Universität Dresden	I
A.2	Process Industries	IV
A.3	Factory Automation	V
	Appendix B Core Language Model	VII
	Appendix C Case Study Results	LXXIX
C.1	Process Industries Visualization Solution	LXXIX
C.2	Factory Automation Visualization Solution	LXXXII
C.3	Energy Supply System Visualization Solution	LXXXIV
C.4	Health Care Visualization Solution	LXXXVII

List of Figures

1.1	Basic Supervisory and Control Paradigm (based on [Sheridan, 1992]).	2
1.2	Illustration of three SCADA applications operated on different platform configurations: (a) shows an embedded device with a native SCADA realization to be exploited close to the process, (b) presents a mobile version for remote access, and (c) illustrates a classical installation of a SCADA application to be operated on a workstation in control rooms. These platform configurations can also be combined.	3
1.3	Basic concept, architecture, and functionality of the proposed MOVISA modeling workbench.	6
2.1	Schematic diagram of the engineering timeline of an <i>Automation System</i> (loosely based on [Urbas and Doherr, 2011]): SCADA engineering takes place at the very end.	10
2.2	A simplified version of the CAMELEON REFERENCE FRAMEWORK [Calvary et al., 2003], as proposed by Limbourg et al. (2005).	13
2.3	Illustration of the core concepts of MDSD: (a) shows the commonly used relation between models; (b) connects these models through transformations.	16
2.4	Comparison of two MDSD compliant modeling techniques: (a) employs the XML technology stack as formal modeling technique; (b) employs the techniques defined in the MDA specification by the OMG.	17
2.5	Artifacts of Domain Specific Languages as formalized by Strembeck and Zdun (2009).	19
3.1	Principle of providing communication drivers for different protocols: A data model organizes different types of process variables; either the data model or the process variable is responsible for converting the data to be used, e.g. by user interface widgets.	34

3.2	Exemplary deployment of a visualization solution: Client-/Server architecture using OPC XML-DA middleware for providing process data also to web clients.	34
3.3	Basic geometric user interface widgets.	35
3.4	Text widgets.	36
3.5	Common interaction widgets (also known from typical office applications).	36
3.6	Interaction widgets that are mandatory for monitoring and operating technical processes.	37
3.7	The root elements of the <i>Core Language Model</i> and their main relationships.	41
3.8	Basic principle of the BOUNDARY concept.	44
3.9	A common information model provides a unified interface to different data provider specific information models.	46
3.10	The LOGICAL DATA PERSPECTIVE represents the common information model, the TECHNICAL DATA PERSPECTIVE captures the data provider specific information models. Both perspectives are connected through an information mapping.	46
3.11	Relationship between a UI COMPONENT and its configurable parameters, classified by the categories <i>Representation</i> , <i>Animation</i> , and <i>Interaction</i>	55
3.12	COMPLEX UI COMPONENT and its properties being defined outside of the component itself.	63
3.13	Navigation path through the displays of a visualization solution. The dashed arrow indicates that “Display 2.1” will be blended in on top of “Display 2”.	64
3.14	Valid example configuration of a READ LINK ACTION.	66
3.15	Valid example configuration of the CLIENT DATA MODEL: constraints ensure a correct information mapping between the logical and technical data perspective.	67
3.16	Valid example configuration of the CLIENT DATA MODEL; language constraints ensure the correct usage of the GENERIC SERVER elements.	68
3.17	Correct (a) and incorrect (b) configuration of a CLONED COMPONENT relationship between two UI COMPONENTS.	68
3.18	Illustrating the need for additional language model constraints: Both relationships “R3” and “R4” must refer to the same object in order to ensure wellformedness of the relationship “R1”.	69

3.19	Illustrating the need for language model constraints: Concrete COMPARATOR types depend on the type of the connected DATA ITEM.	70
3.20	Demonstration of the general language behavior by means of a simple visualization solution at runtime. It can be seen which runtime aspects emerge from the elements of the respective submodel, deduced in Section 3.2.	73
3.21	CLIENT DATA MODEL configuration that ensures the runtime solution always feeding a local data pool with current process values. A special characteristic is that a SUBSCRIPTION can be defined with data items of different data server specifications.	74
3.22	Runtime equivalent of the configuration depicted in Figure 3.21: Although a single subscription was modeled, it results in an appropriate communication stub per data server during runtime.	75
3.23	The DATA ITEM “DI1”, configured in Figure 3.21, defines four specific limit values; an ALARM gives them a certain meaning by stating whether it is an upper or a lower bound alarm.	76
3.24	Runtime behavior of an ALARM element: ALARM BEHAVIORS are represented as mutual exclusive Alarm States; an alarm state in turn can be seen as a state machine.	76
3.25	Example configuration of an ALARM CONTROL widget. Its characteristic ALARM CONTROL ANIMATION property allows for referring to selected alarms that were grouped by particular NOTIFICATION CLASSES. In this example, it only presents the ALARM “A1”, as it is connected to this widget through the element “N” (comp. also Figure 3.23).	77
3.26	Example configuration of a BOUNDARY element intended to integrate application specific code, realizing in this case different methods to convert process data. It expects the data to be converted on the REQUIRED INTERFACE and returns the results through the interface PROVIDED INTERFACE.	78
3.27	Example configuration of a TEXT LABEL widget.	78

3.28	Based on the model depicted in Figure 3.27, a transformation is responsible for making use of the most suitable characteristics: A WRITE DATA ITEM EFFECT tries to realize an atomic write operation if it is provided by the particular data server specification. Otherwise, it uses a fall-back strategy: While the OPC XML-DA specification proposes to write all data items with a single request (comp. Figure 3.29a), a Modbus TCP data provider writes the data items iteratively (comp. Figure 3.29b).	79
3.29	Strategies for writing data items; each strategy is in turn the consequence of the limitations of an individual data server specification.	80
4.1	Improved UML actions, mainly adopted from the symbols of the <i>Scrall</i> language specification [Starr, 2003].	86
4.2	Actions with their symbols that were taken over from the <i>Scrall</i> language specification [Starr, 2003] without modifications. . . .	86
4.3	UML actions with their respective newly created symbols. . . .	87
4.4	Symbols for the actions introduced as extension to the UML specification, as discussed in Section 3.2.1.	88
4.5	Contrasting both the graphical and the textual modeling assets of the PRESENTATION MODEL.	91
4.6	Graphical concrete syntax notation of the PRESENTATION MODEL: The <i>Navigation Subsystem</i> (left hand side) looks very much like a <i>state machine</i> . Each state conceals a <i>Panel Subsystem</i> (right hand side), which provides high-fidelity user interface modeling.	91
5.1	Technology stack to provide model-integrity checks, as proposed by [Raneburger et al., 2011b, Figure 1].	97
5.2	Demonstrating the fundamentals of the used graph algorithms: (a) shows an <i>acyclic directed graph</i> ; (b) presents a <i>cyclic directed graph</i> (solid arrows) as well as the required substitution to become an <i>acyclic directed graph</i> (dashed action and arrows).	101
5.3	Mockup of a dialog dedicated to populate a <i>CUI-to-CUI</i> transformation with platform characteristics.	103
5.4	Sample non-sandboxed deployment with <i>Python</i> as target technology.	105

5.5	Web-based sample deployment with <i>HTML and JavaScript</i> as target technology. Because web applications are executed in a sandbox—the web browser—, certain components need to be transferred to dedicated processing nodes.	106
5.6	The concrete setup to verify the language behavior depends on how the dynamic semantics were defined.	107
5.7	Components of the <i>Movisa</i> modeling workbench.	109
5.8	Movisa Workflow: Modeling, verifying models, and transforming models are separate tasks in the development process.	110
6.1	Excerpt of the PRESENTATION MODEL: The <i>Navigation Subsystem</i> specifies the available PANELS and the NAVIGATION FLOWS between them; the <i>Panel Subsystem</i> hosts the UI COMPONENTS of an individual PANEL (comp. Figure 4.6). Using the NAVIGATION FLOW with the dashed arrow targeting in the “Process Overview Panel”, this target PANEL can be seen as an abstract one which can only be cloned by other PANELS.	115
6.2	Faceplate with a two-stage operation: Using the BUTTONS “Start” and “Stop” is only possible after actuating the “Release” BUTTON.	117
6.3	Excerpt of the ALGORITHM MODEL ensuring to operate the technical process mutual exclusively using Executable UML.	119
6.4	Excerpt of the PRESENTATION MODEL emphasizing on the model annotations as instructions for the <i>CUI-to-CUI</i> transformation: The SIMPLE CONTAINER (❶) component will preserve its contents on big and medium sized screens and will be converted to a TEXT LABEL on small screens. The IMAGE (❷) will only be kept on big screens.	121
6.5	Generated artifacts in relation to the resulting deployment configuration of the process industries case study: A centralized web server operates the <i>Alarm Management</i> and <i>Historical Data</i> components, different visualization clients are equipped with completely generated visualization solutions of different target technologies.	122
6.6	Setup of the factory automation case study.	123

6.7	Resulting deployment configuration of the factory automation case study: The control station “Ifa Nxt Control Ws” controls the production process and provides a web service based interface for visualization clients. Two different runtime solutions were generated from a single MOVISA model.	125
6.8	Demonstration of the basic principle behind the network simulator <i>NetSimP</i> : Each state represents the data model of the entire network in a distinct point in time.	127
6.9	Resulting deployment configuration of the power supply system case study: Runtime solutions of two different target technologies were generated. Both are connected to the network simulator <i>NetSimP</i>	129
6.10	Resulting deployment configuration of the health care case study: Only a <i>non-sandboxed</i> runtime solution is able to meet the requirements.	131
7.1	Classification of the <i>autoHMI</i> concept (white box in the center) by means of the CAMELEON REFERENCE FRAMEWORK (left hand side). The gray boxes present the individual information gathered from the engineering data and stored in the particular models.	138
7.2	Phases of the Useware Engineering process classified by means of the CAMELEON REFERENCE FRAMEWORK. Each development phase comes with a particular modeling language at a different level of abstraction.	140
7.3	Petmap: Reducing developer interaction when applying interactive transformations in iterative development processes (from [Hager et al., 2011]).	141
7.4	Basic principle of the <i>User Centered Design</i> process.	142
7.5	FLEPR: The overall concept showing details for each develop step, namely ① Model Refinement, ② Model Refactoring, and ③ Model Synchronization. This figure also shows which kind of users are involved in the particular development phase.	144

A.1	Mimic of a transformer station in which the power is transformed from medium voltage (from the electricity supplier) to supply voltage (to the consumer). Characteristic in this mimic are the <i>three stage switches</i> : Only after releasing this switch locally, a second release must happen through the visualization solutions before it can be operated.	II
A.2	Mimic of the subsequent hierarchy level: Operative states of the network inside the building can be monitored.	III
A.3	Mimic showing a part of a pharmaceutical process: It is characterized by a static background image augmented with dynamic elements. These dynamic elements reflect the current state of the process by showing actual process values numerically or by means of elements that vary their height (size animations).	IV
A.4	Mimic for monitoring a wafer production line: A robot arm is shown in the center. Hence, user interfaces for monitoring and operating of factory automation processes are characterized by moving elements (position animations).	V
B.1	Core Language Model: Movisa Root.	VII
B.2	Core Language Model: Algorithm Root.	VIII
B.3	Core Language Model: Class Subsystem.	IX
B.4	Core Language Model: Boundary Subsystem.	X
B.5	Core Language Model: Data Type Subsystem.	XI
B.6	Core Language Model: State Machine Subsystem.	XII
B.7	Core Language Model: Action Root Subsystem.	XIII
B.8	Core Language Model: Action Pin Flow Subsystem.	XIV
B.9	Core Language Model: Action Collection.	XV
B.10	Core Language Model: Variable Action Subsystem.	XVI
B.11	Core Language Model: Accept Event Action Subsystem.	XVII
B.12	Core Language Model: Send Signal Action Subsystem.	XVIII
B.13	Core Language Model: Object Action Subsystem.	XIX
B.14	Core Language Model: Read Boundary Action Subsystem.	XX
B.15	Core Language Model: Read Presentation Model Action Subsystem.	XXI
B.16	Core Language Model: Value Specification Action Subsystem.	XXII
B.17	Core Language Model: Structural Feature Action Subsystem.	XXIII
B.18	Core Language Model: Data Item Action Subsystem.	XXIV
B.19	Core Language Model: Link Action Subsystem.	XXV

B.20 Core Language Model: Expansion Region Action Subsystem. . .	XXVI
B.21 Core Language Model: Presentation Model Root.	XXVII
B.22 Core Language Model: Navigation Subsystem.	XXVIII
B.23 Core Language Model: Multi Lingual Text Definition Subsystem.	XXIX
B.24 Core Language Model: Image Bundle Subsystem.	XXX
B.25 Core Language Model: Color Definition Subsystem.	XXXI
B.26 Core Language Model: Representation Record Subsystem. . . .	XXXII
B.27 Core Language Model: Representation Subsystem (1).	XXXIII
B.28 Core Language Model: Representation Subsystem (2).	XXXIV
B.29 Core Language Model: Representation Concrete Type Subsystem.	XXXV
B.30 Core Language Model: Representation Scale Subsystem.	XXXVI
B.31 Core Language Model: Representation Indicator Subsystem. . .	XXXVII
B.32 Core Language Model: Animation Subsystem (1).	XXXVIII
B.33 Core Language Model: Animation Subsystem (2).	XXXIX
B.34 Core Language Model: Animation Subsystem (3).	XL
B.35 Core Language Model: Animation Subsystem (4).	XLI
B.36 Core Language Model: Interaction Subsystem.	XLII
B.37 Core Language Model: Interaction Subsystem (Key Code Con- stants).	XLIII
B.38 Core Language Model: Interaction Effect Subsystem (1).	XLIV
B.39 Core Language Model: Interaction Effect Subsystem (2).	XLV
B.40 Core Language Model: Elementary UI Component Root.	XLVI
B.41 Core Language Model: Complex UI Component Root.	XLVII
B.42 Core Language Model: Alarm Control Widget.	XLVIII
B.43 Core Language Model: Button Widget.	XLIX
B.44 Core Language Model: Check Box Widget.	L
B.45 Core Language Model: Drop Down Widget.	LI
B.46 Core Language Model: Ellipse Geometrical Object.	LII
B.47 Core Language Model: Gauge Widget.	LIII
B.48 Core Language Model: Image Widget.	LIV
B.49 Core Language Model: Input Widget.	LV
B.50 Core Language Model: Polyline Geometrical Object.	LVI
B.51 Core Language Model: Polygon Geometrical Object.	LVII
B.52 Core Language Model: Radio Button Widget.	LVIII
B.53 Core Language Model: Slider Widget.	LIX
B.54 Core Language Model: Table Widget (Structure).	LX
B.55 Core Language Model: Table Widget (Properties).	LXI
B.56 Core Language Model: Text Label Widget.	XLII

B.57	Core Language Model: Tree Widget (Structure).	LXIII
B.58	Core Language Model: Tree Widget (Properties).	LXIV
B.59	Core Language Model: Trend Widget.	LXV
B.60	Core Language Model: Client Data Model Root.	LXVI
B.61	Core Language Model: Logical Data Perspective.	LXVII
B.62	Core Language Model: Alarm Perspective (1).	LXVIII
B.63	Core Language Model: Alarm Perspective (2).	LXIX
B.64	Core Language Model: Technical Data Perspective Root.	LXX
B.65	Core Language Model: Technical Data Perspective (OPC XML-DA).	LXXI
B.66	Core Language Model: Technical Data Perspective (Modbus TCP).	LXXII
B.67	Core Language Model: Technical Data Perspective (OPC UA).	LXXIII
B.68	Core Language Model: Technical Data Perspective (OPC UA, Data Types 1).	LXXIV
B.69	Core Language Model: Technical Data Perspective (OPC UA, Data Types 2).	LXXV
B.70	Core Language Model: Technical Data Perspective (IfaNxtControlWs).	LXXVI
B.71	Core Language Model: Technical Data Perspective (Generic Server).	LXXVII
C.1	Process Industries Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). It can be seen that the operating faceplate is blocked. To demonstrate multilingualism, the user interface language was switched to <i>German</i> .	LXXIX
C.2	Process Industries Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). It can be seen that the operating faceplate is released and, thus, interventions in the process are allowed.	LXXX
C.3	Process Industries Case Study: Generated <i>HTML</i> based runtime solution (sandboxed). It can be seen that the operating faceplate is blocked. Additionally, the button to request the operation token is also blocked due to interventions by other operators (e.g. through the visualization solution shown in Figure C.2). To demonstrate multilingualism, the user interface language was switched to <i>English</i> .	LXXX

C.4	Process Industries Case Study: Generated <i>HTML</i> based runtime solution (sandboxed). It can be seen that the operating faceplate is released and, thus, interventions in the process are allowed.	LXXXI
C.5	Process Industries Case Study: After applying a horizontal CUI-to-CUI transformation and a subsequent vertical CUI-2-FUI transformation, the resulting solution can be used on the iPhone. These screenshots show the solution without manually improving the model after translating it into this new context of use.	LXXXI
C.6	Factory Automation Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed).	LXXXII
C.7	Factory Automation Case Study: Generated <i>HTML</i> based runtime solution (sandboxed).	LXXXIII
C.8	Energy Supply System Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). This figure presents the topmost hierarchy level of the energy supply network. It can be seen that alarms were thrown indicating that the network has a too high load.	LXXXIV
C.9	Energy Supply System Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). This figure presents a view on the energy supply network of “Building A”. It can be seen that the reason for the alarms is located in “Unit 2”.	LXXXIV
C.10	Energy Supply System Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). This figure presents a view on the energy supply network of “Unit 2” in “Building A”. The reason for the alarms could be identified and eliminated. Hence, the alarms are no longer in <i>active</i> state, even though they are <i>not acknowledged</i>	LXXXV
C.11	Energy Supply System Case Study: Generated <i>HTML</i> based runtime solution (sandboxed). This figure presents the topmost hierarchy level of the energy supply network. It can be seen that alarms were thrown indicating that the network has a too high load.	LXXXV
C.12	Energy Supply System Case Study: Generated <i>HTML</i> based runtime solution (sandboxed). This figure presents a view on the energy supply network of “Building A”. It can be seen that the reason for the alarms is located in “Unit 2”.	LXXXVI

C.13 Energy Supply System Case Study: Generated <i>HTML</i> based runtime solution (sandboxed). This figure presents a view on the energy supply network of “Unit 2” in “Building A”.The reason for the alarms could be identified and eliminated. Hence, the alarms are no longer in <i>active</i> state, even though they are <i>not acknowledged</i>	LXXXVI
C.14 Health Care Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). Patients has to answer questions about their recent habits.	LXXXVII
C.15 Health Care Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). First, a patient decided to fill in the number of tablets recently consumed.	LXXXVIII
C.16 Health Care Case Study: Generated <i>Python</i> based runtime solution (non-sandboxed). After providing the answer about medication, this option disappeared from the screen and, thus, it cannot be selected no longer.	LXXXIX

List of Tables

3.1	Classes of common properties that were identified during investigating visualization systems as well as running solutions.	38
3.2	Various possibilities to alter widgets.	39
3.3	Configurable interaction effects.	40
3.4	Characteristic parameters of different data server specifications.	47
3.5	Characteristic parameters to be considered for the definition of Alarms.	51
3.6	Aspects to be considered when presenting alarms to operators.	52
3.7	Different kinds of animation properties for a precise specification a UI COMPONENTS behavior.	57
3.8	Interaction properties for modeling UI COMPONENTS which are sensible to interactions of human operators.	58
3.9	ELEMENTARY UI COMPONENTS with their specific characteristics.	60
3.10	Complex UI Components.	62
5.1	Supported model annotation properties.	103

List of Listings

4.1	C struct.	88
4.2	Java class.	88
4.3	General construction rule of the text based concrete syntax notation.	89
4.4	Example configuration of the LOGICAL DATA PERSPECTIVE using the concrete syntax notation.	89
6.1	TECHNICAL DATA PERSPECTIVE	114
6.2	LOGICAL DATA PERSPECTIVE	114
6.3	ALARM PERSPECTIVE	114
6.4	Setting a local DATA ITEM through a BUTTON's interaction property.	116
6.5	Altering a BUTTON's ACCESSIBILITY property.	116
6.6	BUTTON INTERACTION for realizing an <i>atomic write</i> operation. . .	118
6.7	Defining the alarms to be presented in a particular ALARM CONTROL widget.	118
6.8	Platform specific BOUNDARY realization ensuring to send a SIGNAL after a certain time slot expired.	119
6.9	Excerpt of the TECHNICAL DATA PERSPECTIVE showing the configuration of the newly integrated data provider.	124
6.10	Excerpt of the TECHNICAL DATA PERSPECTIVE showing a configuration of the sample network presented in Figure 6.8 through the GENERIC SERVER terminology.	128

Nomenclature

API	Application Programming Interface
ASL	Action Specification Language
ATL	ATLAS Transformation Language
AUI	Abstract User Interface
CAE	Computer Aided Engineering
CAEX	Computer Aided Engineering eXchange
CRF	Cameleon Reference Framework
CUI	Concrete User Interface
DES	Discrete Event System
DSL	Domain Specific Language
EBNF	Extended Backus-Naur Form
EGL	Epsilon Generation Language
EOL	Epsilon Object Language
EVL	Epsilon Validation Language
FCML	Facility Control Markup Language
FDA	U.S. Food and Drug Administration
FLEPR	Flexible Workflow for early User Interface Prototypes
FUI	Final User Interface
FUML	Semantics of a Foundational Subset for Executable UML Models
GUI	Graphical User Interface
HCI	Human Computer Interaction
HMI	Human Machine Interface
HTML	Hypertext Transfer Markup Language
JET	Java Emitter Templates
M2M	Model to Model (Transformation)
M2T	Model to Text (Transformation)
MARIA	Model-based lAnguage foR Interactive Applications
MBUID	Model Based User Interface Development
MDA	Model Driven Architecture
MDSD	Model Driven Software Development
MOF	Meta Object Facility
OAL	Object Action Language

OCL	Object Constraint Language
OMG	Object Management Group
OPC	OLE for Process Control (<i>Nowadays, OPC is used without referring to an abbreviation, as the importance of the OLE interface decreases.</i>)
OPC UA	OPC Unified Architecture
PLC	Programmable Logic Controller
PUC	Personal Universal Controller
QVT	Query/Views/Transformations
SCADA	Supervisory Control and Data Acquisition
Scrall	Starr's Concise Relational Action Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVG	Scalable Vector Graphics
T&C	Tasks and Concepts
UCD	User Centered Design
UI	User Interface
UIML	User Interface Markup Language
UML	Unified Modeling Language
UsiXML	User Interface Markup Language
VBA	Visual Basic for Applications
WSDL	Web Services Description Language
XIML	eXtensible Interface Markup Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation
XUL	XML User Interface Language
XVCML	eXtensible Visualization Components Markup Language

Chapter 1

Introduction

Automation systems consist of technical processes and the required automation equipment [Lunze, 2008]. A technical process is, according to Johannsen (1993), a physical-technical or a chemical-technical procedure with material, energy, and/or information flows at input and output. Johannsen (1993) instanced a procedure on a milling machine with a raw workpiece and electrical energy as input, the processed workpiece, chips, and thermal energy as output. The automation equipment is composed of devices required to transform material and energy. These devices impact on the technical process, e.g. a servo motor moving the milling head.

Human operators are responsible for a safe operation of an automation system [Johannsen, 1993]. Hence, it is monitored and operated by human operators through appropriate *Human Machine Interfaces* (HMI), constituting the connection between the human operator and the automation system: It allows for monitoring the operative states of the automation system by presenting relevant information that have appropriately been prepared. Furthermore, it consists of input devices for entering information. In this way, human operators are enabled to have an impact on the technical process according to its actual state and given goals. Sheridan (1992) makes a distinction between *Human Computer Interaction* (HCI) and *Supervisory Control*: While in HCI one uses computers to operate other computers or databases as end objects, in *Supervisory Control* computers are only mediators between a technical process and human supervision. Sheridan (1992) defines *Supervisory Control* as follows: “[...] one or more human operators are intermittently programming and continually receiving information from a computer that itself closes an autonomous control loop through artificial effectors and sensors to the controlled process or task environment”¹. Figure 1.1 shows the basic paradigm behind this definition. Appendix A exemplarily presents concrete supervisory control solutions.

¹Cassandras and Lafortune (1999) introduce *Supervisory Control* in automata theory for a

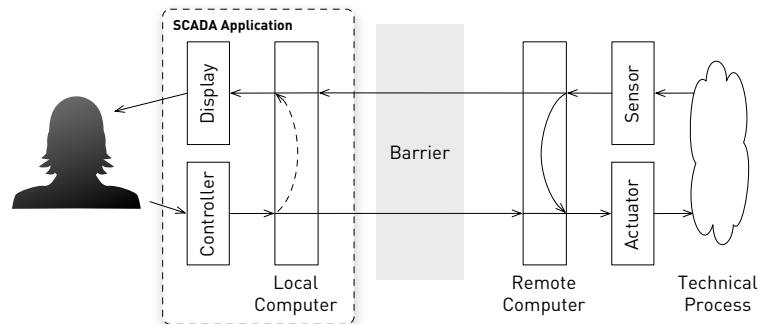


Figure 1.1: Basic Supervisory and Control Paradigm (based on [Sheridan, 1992]).

A remote computer closes a control loop by observing values actually measured by sensors and by setting particular actuators according to these values and given (control) programs. Human operators supervise this process through *graphical displays* representing the technical instrumentation and employ appropriate controllers to modify parameters in the remote programs. Both, the operator and the remote device, might be separated by “a barrier of distance, time, or inconvenience” [Sheridan, 1992]. To implement this general concept, *Programmable Logic Controllers* (PLC) that act independently and close to the process are usually used as remote computers. *Supervisory Control and Data Acquisition* (SCADA) applications constitute the local part depicted in Figure 1.1. Bailey and Wright (2003) refer SCADA “to the combination of telemetry and data acquisition”: It encompasses the collection of relevant information, carrying out any necessary analysis and preparing that information to be presented on a number of operator screens or displays. Required control actions are then conveyed back to the process. A SCADA application is, according to Daneels and Salter (1999), a purely software package that is positioned on top of hardware. Thus, SCADA applications form the *Human Machine Interface* for process visualization.

Definition 1.1: A Visualization System is a software tool to develop SCADA applications. A Visualization Solution is a particular SCADA application tailored

given *Discrete Event System* (DES) “whose behavior must be modified by feedback control in order to achieve a given set of specification.” This DES is modeled by a graph G with an event set E . If the behavior of G is not satisfactory, it must be controlled by a supervisor S . S observes all events that G executes and then, “ S tells G which events in the current active event of G are allowed next.”

to supervise a concrete technical process, thus being the User Interface to this process.

In the following, Section 1.1 explains the necessity for a new approach to the development of visualization solutions and introduces the solution proposed in this thesis. Section 1.2 provides an overview of structure of this thesis.

1.1 Problem Statement and Aims

Vital requirements for graphical human machine interfaces for process control are among others *efficiency, ergonomics*, and that they “shall be so designed as to allow the operator to perform [her] activities in accordance with [her] capabilities, skills and needs, as required to achieve [her] objectives” [VDI/VDE, 2005]. For this purpose, the guideline VDI/VDE (2005) recommends to involve operators into the design phase. Therewith it proposes a *User Centered Design (UCD)* approach, thus entailing a significant amount of the overall system design and development effort. On the other hand, platforms executing SCADA applications in industrial automation are characterized by diversity as illustrated in Figure 1.2.

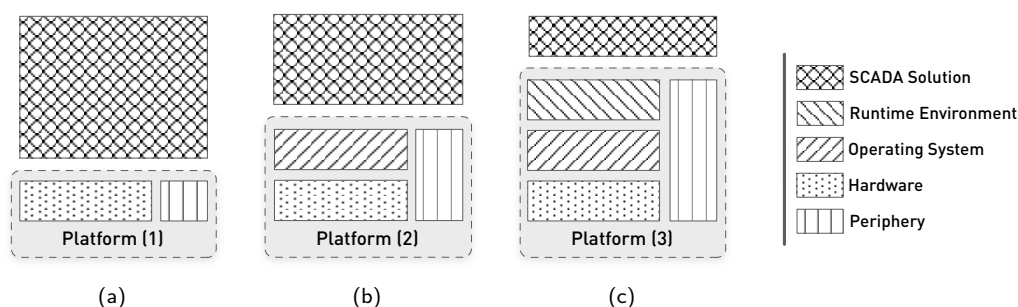


Figure 1.2: Illustration of three SCADA applications operated on different platform configurations: (a) shows an embedded device with a native SCADA realization to be exploited close to the process, (b) presents a mobile version for remote access, and (c) illustrates a classical installation of a SCADA application to be operated on a workstation in control rooms. These platform configurations can also be combined.

Currently, visualization solutions are developed using platform specific tools and terminologies, such as a specific programming language. As a consequence, operative characteristics are strongly merged with aspects of their technical realization. Thus, a human machine interface has to be redeveloped for and

tested on each platform using different and probably incompatible tools, even if the operative characteristics remain the same. Moreover, as Menzel et al. (2003) state, the situation is exacerbated by the fact that standard end-user platforms are used almost exclusively. These platforms are characterized by short innovation cycles decreasing the life time of SCADA components to less than five years due to regular releases of new software versions [Menzel et al., 2003]. Given the fact that industrial facilities are required to be in operation for decades and given the complexity of SCADA applications in order to ensure correctness, reliability, and usability, new design approaches are demanded to suit the rapid development of the platforms.

Model driven techniques are promising design approaches, as they express operative characteristics through a platform independent terminology that is based on models. Technical aspects are separated into transformations as carriers of platform specific terminologies. For the given situation, model driven approaches enable to capture the operative characteristics of a SCADA solution by creating a platform independent model. Deploying this solution to a particular platform requires an appropriate transformation that translates the platform independent terminology, provided by the model, into a platform specific terminology. If a compatible platform evolves over time, e.g. through an update of the operating system, causing incompatibility or to equip another or a new platform with this existing SCADA solution, only suitable transformation is required while the respective model remains unchanged. Consequently, operational characteristics need to be expressed and tested for correctness, reliability, and usability only once. Tested and correct transformations are expected to produce always correct platform specific runtime solutions. Additionally, model driven approaches offer the following more general benefits:

- (1) Functional aspects can be reused in future projects, even if they aim at another platform.
- (2) Technical realizations can be reused in future projects, even if the functionality is a different one.
- (3) High-quality and reproducible solutions can be generated through tested transformations.
- (4) Various technical solutions can automatically be created from a single functional description, simply by invoking another transformation.
- (5) The repertoire of required tools can be slimmed down.

Nichols, Chau, and Myers (2007) have already proven the viability of model driven approaches for the development of user interfaces for office applications. Aquino et al. (2010) additionally proved that model driven development

procedures are promising even for automatically deploying user interfaces to different devices. Visualization solutions in industrial environments, however, are characterized by specific requirements and constraints that the aforementioned approaches to model office solutions do not meet, mainly due to the fact that office applications are connected to databases, whereas industrial solutions are connected to technical processes.

This thesis transfers the aforementioned advantages of model-driven approaches into the domain of industrial *production automation*². It proposes a *Domain Specific Language* which enables the development of sustainable visualization solutions. Other than model-driven approaches for office applications, this modeling language is tailored to meet the requirements of industrial automation: (1) It defines a sufficient set user interface components with appropriate *representation*, *animation*, and *interaction* properties. (2) It provides a solid abstraction to the variety of industrial automation specific process communication means. (3) It contains an *Executable UML* realization to take into account the diversity of industrial processes and their individual requirements.

This thesis contributes the domain specific modeling workbench MOVISA, as depicted in Figure 1.3: MOVISA models capture operative characteristics of visualization solutions. Transformation rules capture aspects of the technical realization. The verification tool ensures the correctness of models and the transformation tool processes MOVISA models either to modify these models or to automatically create runtime artifacts. This complexity is encapsulated behind a high-fidelity modeling interface to be exploited by domain experts. Engineering tools are able to populate MOVISA models through low-fidelity tool interface.

²Johannsen (1993) brings the fields of *process industries*, *factory automation*, and *energy supply systems* under the umbrella of production automation together.

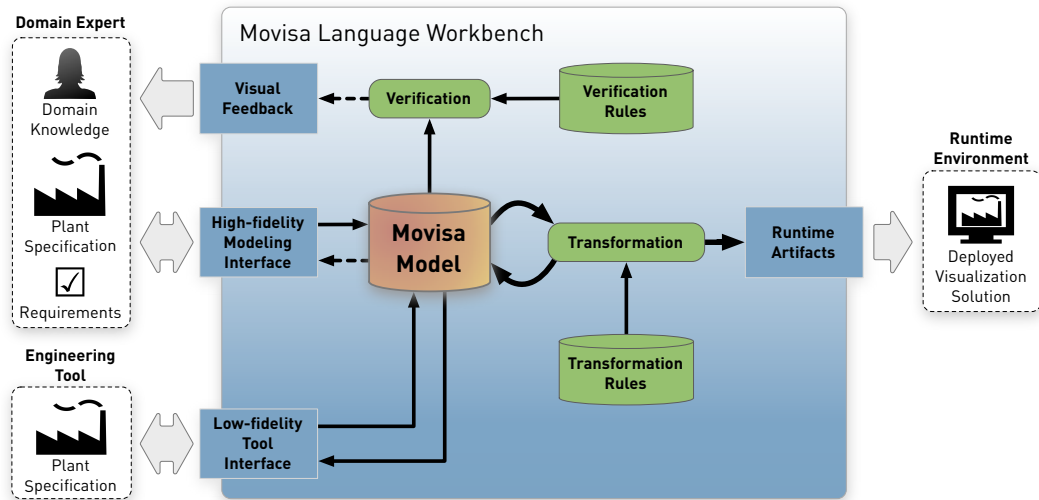


Figure 1.3: Basic concept, architecture, and functionality of the proposed MOVISA modeling workbench.

1.2 Thesis Structure

This thesis is structured as follows:

Chapter 2 gives an overview of the state of the art and its particular background. This chapter concludes with a detailed requirements definition used for pointing out deficiencies of the state of the art approaches and how this thesis contributes to them.

Chapter 3 formalizes the *Language Model* of the *Domain Specific Language* MOVISA by abstracting its *Target Domain* into the *Core Language Model*. Section 3.3 explains *Language Constraints* and Section 3.4 presents the *Language Behavior* definition.

Chapter 4 works out and discusses a concrete syntax notation that enables modelers to work and to think their domain.

Chapter 5 discusses required aspects of a modeling workbench for creating, using, and maintaining models of the language that was created in Chapter 3 which encapsulates the complexity of the model driven approach.

Chapter 6 evaluates the feasibility of the MOVISA modeling workbench by exploiting it on representative case studies.

Chapter 7 elaborates a transformation based framework enabling to incorporate the MOVISA modeling workbench in higher-level engineering procedures.

Chapter 8 draws the conclusions of the findings of the previous chapters.